Toni Harju

Grafiikkamoottori

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

30.4.2015

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Toni Harju Grafiikkamoottori 44 sivua 30.4.2015
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Lehtori Miikka Mäki-Uuro Päätoiminen tuntiopettaja Heini Puuska

Insinöörityössä käsitellään tietokonegrafiikkaan ja ohjelmistotekniikkaan liittyvää termiä grafiikkamoottori. Työn alussa määritellään käsitettä tarkemmin analysoimalla ohjelmisto-teknistä arkkitehtuuria, yleisiä sovellustarkoituksia ja arvioimalla lyhyesti liiketaloudellisia tekijöitä.

Työssä edetään käsittelemään grafiikkamoottoriin toteutukseen liittyviä ongelmallisia ky-symyksiä, teknologioita, tekniikoita sekä toteutuksia.

Työhön kuuluu oman grafiikkamoottorin toteutuksen dokumentointi liittyen projektin ideaan, suunnitteluun, hallintaan, ohjelmistoteknilliseen arkkitehtuuriin komponentteihin, käytettyi-hin tekniikoihin ja teknologioihin sekä jatkosuunnitelmat projektille.

Työn lopputuloksena on tämä dokumentti sekä grafiikkamoottori, joka toimii runkona tai alustana tulevaisuuden projekteille.

Avainsanat	Grafiikkamoottori, OpenGL, Lua

Author(s) Title Number of Pages Date	Toni Harju Graphics Engine 44 pages 30 April 2015
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Miikka Mäki-Uuro, Senior Lecturer Heini Puuska, Lecturer

The thesis discusses the term graphics engine that is related to computer graphics and software engineering. The study starts with determining the term with precise methods that include software architecture, common implementations and analysis of business benefits and disadvantages.

The study continues to discuss the problems, technologies, techniques and implementations related to graphics engine.

The study includes the documentation related to the author's implementation of a graphics engine. The documentation contains the information on the idea of the project, as well as the planning, management, software architectural components, technique, technologies and future plans for the project.

The result of the study in addition to this documentation was the implantation of a graphics engine that will be used as a framework or platform for future projects.

Keywords	Graphics engine, OpenGL, Lua

Sisällys

Termistö

1 Johdanto 1

2 Käsite 1

2.1 Määritelmä 1
2.2 Tyypit 2
2.3 Sovellustarkoituksia 3
2.4 Liiketaloudellisia vaikutuksia 4

3 Toteutus 4

3.1 Abstraktiotasot 4
3.2 Keskusyksikkö vai näytönohjain 5
3.3 Näytönohjainrajapinnat 6
3.3.1 OpenGL 6
3.3.2 Direct3D 7
3.3.3 Yksinkertaistettu OpenGL ja Direct3D -liukuhihna 8
3.3.4 OpenCL 10
3.3.5 NVIDIA CUDA 11

4 Realistisuus ja reaaliaikaisuus 12

4.1 Laskutehokkaat tekniikat 12
4.1.1 Phongin sävytys- ja heijastusmallit 12
4.1.2 Kohoumakuvaus 13
4.1.3 Normaalikuvaus 14
4.1.4 Varjokuvaus 14
4.1.5 Taustavalaistuksen okkluusio 15
4.1.6 MIP-kuvaus 16
4.2 Fotorealistiset tekniikat 17
4.2.1 Säteenjäljitys 17
4.2.2 Radiositeetti 18

5 Toteutuksia 19

5.1 OGRE 19
5.2 Horde3D 22
5.3 POV-Ray 24

6 Oma toteutus 25

6.1 Idea 25
6.2 Projektin rajaus 26
6.3 Tutkimusvaihe 27
6.4 Projektin hallinta 27
6.4.1 Projektin kehitysympäristö 27
6.4.2 Projektin ylläpito ja seuranta 28
6.5 Ohjelma-arkkitehtuurin komponenttien suunnittelu 28
6.5.1 Ohjelmointikieli 28
6.5.2 Skriptikieli 29
6.5.3 Ohjelmointirajapinta 29
6.5.4 Ikkunointi 31
6.5.5 Resurssien lataaminen 32
6.5.6 Resurssien hallinta 33
6.5.7 Näkyvyyden hallinta 34
6.5.8 Kuvan piirtäminen 34
6.5.9 Kommunikaatio näytönohjaimen kanssa 35
6.6 Sovelluksen matematiikka ja algoritmit 36

7 Jatkokehityssuunnitelmat 37

8 Oppimien asioiden analysointi ja arviointi 38

Lähteet 40

Termistö

Geometria	Matemaattisesti määritelty muoto, jolla on suhteellinen koko ja sijainti.
Renderöinti	Geometrisistä primitiiveistä visualisoidaan digitaalisesti kuva tietokoneohjelman avulla tiedostoon tai laitteiston muistiin.
Verteksi	Geometrisen primitiivin kulmapiste.
Tekstuuri	Pintakuviointi geometrialle, joka yleisesti tallennetaan kuvaformaatissa.
Sävytinohjelma	(Shader engl.) Ohjelma, joka lähetetään näytönohjaimeen käännettäväksi, jolla voidaan vaikuttaa näytönohjaimen liukuhihnan toimintaan.
Verteksioperaatio	Geometrioiden kulmapisteiden, vektorien ja matriisien manipuloimista kuvan oikeanlaisen perspektiiviin ja projektion luomiseksi.
Pikselioperaatio	Geometrioiden pintojen tai näkymän leikattujen fragmenttien sävyttämiseen tarkoitettu toiminto, jossa tekstuurien tai muiden värilähteiden arvoja manipuloidaan, jolloin lopullinen pikseli saa halutun arvon.
Liukuhihna	Näytönohjaimen tai näytönohjaimen rajapinnan abstrahoima prosessi, jossa tehdään toiminnallisuuksia sarjassa

1 Johdanto

Olen ollut hyvin kiinnostunut tietokonegrafiikasta ja informaation visualisoinnista pienestä lapsesta lähtien. Erilaisten pelien tietokonegrafiikat ja tietokoneella visualisoidut kuvat, animaatiot ja elokuvat ovat inspiroineet elämäni tekemisiä ja tuottaneet mieltä stimuloivia kokemuksia ja herättäneet suurta uteliaisuutta niiden teknistä toteutusta kohtaan. Oma uteliaisuus on ajan myötä vain vahvistunut ja vihdoin tämän projektin kautta minulla on elämässäni viipale aikaa keskittyä tutkimaan tarkemmin hyvinkin haasteellisia tietokonegrafiikan toteutukseen liittyviä algoritmeja, tekniikoita, teknologioita ja ohjelmistoarkkitehtuurillisia kysymyksiä.

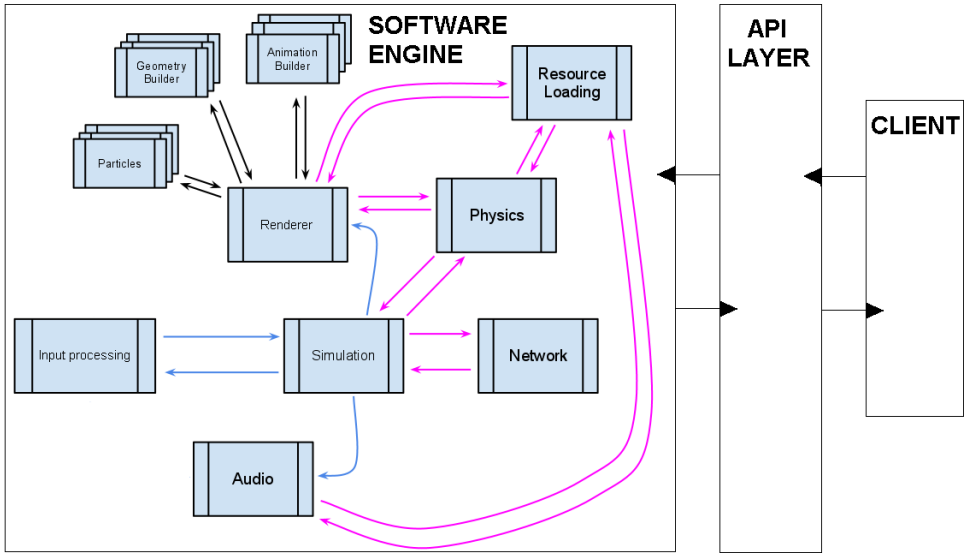
Toivon tämän työn avulla selkeyttäväni grafiikkamoottori käsitettä ja saavani sen merkityksen yleisempää tietoisuuteen ja mahdollisesti inspiroimaan ja motivoimaan jonkun innostumaan tietokonegrafiikkaan liittyvistä aiheista samalla tapaa kuten itse olen aikaisemmin inspiroitunut. Suunnittelen tekeväni tulevaisuudessa tietokonegrafiikkaan liittyviä projekteja harrastusten, opiskelun tai työn parissa, kun projektin myötä olen saanut lisää informaatiota ja kokemusta aiheeseen liittyen.

2 Käsite

2.1 Määritelmä

Moottori-käsitteellä tarkoitetaan ohjelmistoarkkitehtuurin osa-aluetta, joka automatisoi, toimii modulaarisesti ja abstrahoi esimerkiksi muistinhallintaa, tietoliikenneyhteyksiä, grafiikkaa, animaatiota, fysiikkaa, äänen käsittelyä, tekoälyä, komentosarjakielen toiminnollisuutta tai lokalisointia. Kuviossa 1. on esitetty pelkistetty malli. (1; 2.)

Käsitettä voidaan hyvin verrata arkielämän tapahtumaan; autoon ja sen kuljettajaan. Kuljettaja ei tiedä tarkalleen, kuinka auton eri komponentit toimivat yhdessä, mutta silti hän pystyy liikkumaan paikasta A paikkaan B abstrahoidun rajapinnan ja informaation avulla.



Kuvio 1. Yksinkertaisen moottorin mallinnus

Tietokonegrafiikka on tietojenkäsittelyopin tai tietotekniikan alalaji, jossa tutkitaan tapoja syntetisoida ja manipuloida visuaalista sisältöä (kuvia, animaatioita, 3d-grafiikkaa, videoita). (3.)

Grafiikkamoottorin perimmäinen tarkoitus on edesauttaa loppukehittäjiä tai asiakasohjelmia visuaalisten sovelluksien kehittämisessä luomalla joustava, dynaaminen ja modulaarinen rajapinta tai runko. Rungon tai rajapinnan tulee pystyä toimimaan täysin itsenäisesti vain ulkoisten tietovirtojen avulla. (1; 3; 2.)

Grafiikkamoottorin päätehtäviin kuuluu hallita visuaalista näkyvyyttä luomalla kuva käyttäjän antamista tietorakenteista tai komennoista. Tietorakenteet tai komennot sisältävät informaatiota geometrioista, perspektiivistä, tekstuureista, valaistuksesta sekä muista graafisista efekteistä ja ominaisuuksista. Kehittäjät voivat keskittyä sovelluksen funktionaalisiin ongelmiin syventymättä alempien tasojen teknillisiin haasteisiin.

2.2 Tyypit

Grafiikkamoottorit voidaan karkeasti jakaa kahteen osajoukkoon reaaliaikaisiin ja aikariippumattomiin sovelluksiin.

Aikariippumattomat eli esirenderöityt sovellukset käyttävät tietokonekuvan luomiseen laskenta-aikaa vaativia tekniikoita kuten säteenjäljitys tai joku toinen globaali valaistusmalli, joilla kuviin saada hyvinkin fysikaalisesti realistisia ominaisuuksia. Aikavaativuudesta ja informaatiomäärästä johtuen kuvat yleensä tallennetaan tietokoneen kovalevylle kuvatiedostoina. (3.)

Reaaliaikaisissa sovelluksissa käytetään kuvien muodostamiseen laskentatehokkaita malleja, jotka pyrkivät realistisiin ratkaisuihin ja kuvaa ei yleisesti tallenneta kovalevylle, vaan sitä päivitetään jatkuvasti tietokoneen tai näytönohjaimen muistissa useita kymmeniä kertoja sekunnissa, jolloin luodaan illuusio liikkuvasta visuaalisesta informaatiosta. (4.)

2.3 Sovellustarkoituksia

Grafiikkamoottoreita voidaan soveltaa teollisuuden alan tarkoituksiin. Pelialalla grafiikkamoottoreita käytetään osana pelimoottoria pelitilojen reaaliaikaiseen visualisointiin. Pelin hetkittäisestä tilasta visualisoidaan kuva, jonka avulla pelaajan on helpompi ymmärtää pelin kulkua ja informaatiota. (5.)

Elokuva-alalla grafiikkamoottoreita käytetään elokuvien editoimiseen ja renderöimiseen. Renderöidessä videolaitteella tallennettujen kuvien väri-informaatioden avulla luodaan uusia kuvia erilaisilla tekniikoilla, jotka voivat olla säteen jäljitystä, fysikaalisten mallien simulointia tai erilaisia approksimoivia malleja. (6.)

Grafiikkamoottori soveltuu hyvin myös eri tieteenalojen tutkimus- ja havainnointiprosesseihin, joissa informaatiota visualisoidaan ihmisen ymmärrettävään muotoon. (7.)

Grafiikkamoottoria voidaan käyttää komponenttina virtuaalitodellisuuden luomiseen. Virtuaalitodellisuus tarkoittaa virtuaalisesti simuloitua todellisuutta, jossa grafiikkamoottori sovellettaisiin virtuaalimaailman visuaalisten ärsykkeiden luomiseen. (8.)

Grafiikkamoottoria voidaan soveltaa laajennettuun todellisuuteen (engl. augmented reality), jossa läpikatseltaviin näyttöihin lisätään näköaistin luomaan todelliseen ympäristöön tietokoneella tuotettua tietoa (kuvia, videota, tekstiä tai GPS-informaatiota).

(9.)

2.4 Liiketaloudellisia vaikutuksia

Välittömät hyödyt grafiikkamoottorin käytössä kohdistuvat kehitys- ja ylläpitokustannuksiin. Jokaisen asiakasohjelman kehittäjän ei tarvitse kouluttautua matalan abstraktion, teknisen ja hyvin spesifisen toimialan pariin. Asiakasohjelman arkkitehdin on helppo integroida autonominen komponentti järjestelmään ilman muita riippuvuuksia. Asiakasohjelman ylläpitäjien ei tarvitse huolehtia ohjelmavirheiden korjaamisesta, vaan grafiikkamoottorin kehittäjät ottavat vastuun niiden korjaamisesta. Asiakasohjelman markkinointi voi helpottua, jos grafiikkamoottori itsessään on saavuttanut hyvin brändätyn imagon.

Informaation käsittely on ulkoistettu suljettuun komponenttiin, jonka lähdekoodiin tai toteutuksen muuttaminen on käytännössä erittäin vaikeaa. Se voi pahimmillaan luoda kehitystilanteisiin tilanteita, joista ei päästä eteenpäin konsultoimatta tai toteuttamalla komponenttia itse, vaihtamalla sovellusta toiseen tai odottamatta grafiikkamoottorin kehittäjien korjausta virhetilanteeseen.

3 Toteutus

3.1 Abstraktiotasot

Abstraktiotasolla vastataan grafiikkamoottorin yhteydessä kysymyksiin; onko ohjelma täysin laitteistoriippuvainen tai onko se hyvin abstrahoitu kaikenlaisia käyttöjärjestelmiä, kirjastoja tai rajapintoja vasten.

Matalamman abstraktiotason grafiikkamoottori on yleensä näytönohjaimen valmistajan mikropiirejä hallinnoiva prosessi, joka on ohjelmoitu kuuntelemaan ulkoisilta rajapinnoilta tulevia komentoja. Rajoittavina tekijöinä ovat fyysiset komponentit, laitearkkitehtuuri, käskykanta ja tekniset tietovaatimukset. Tällä toteutustavalla saadaan laitteistosta suoraan kaikki tehot irti, ja kaikki riippuvuudet ovat todella yksinkertaisia.

Edellä mainittujen rajoittavien tekijöistä kehittyvistä ominaisuuksista tai toiminnallisuuksista johtuen, on realistista odottaa korkeintaan muutamaa erilaista toteutusta grafiikkamoottorin toiminnoille, jotka abstrahoidaan korkeamman tason rajapinnan avulla käyttäjille. Muutoin grafiikkamoottorin kehittäjät joutuvat muuttamaan koodinsa täysin laitteesta riippuen, joka ei vastaa kestävän kehityksen kriteereitä eikä ole muutenkaan tehokasta resurssien käyttöä.

Korkeamman abstraktitason grafiikkamoottori voidaan toteuttaa täysin käyttöjärjestelmän tai muun prosessoivan laskenta-alustan kautta (CUDA:n näytönohjaimet tai OpenCL-prosessorit). Laskutoimituksia toteutusta voidaan kiihdyttää keskusyksikön, näytönohjaimen tai prosessoivan mikropiirin avulla näytönohjaimen tai laskenta-alustan tarjoaman rajapinnan kautta. Silloin voidaan saavuttaa monen eri laitteen ja ohjelman hyödyt. Haittapuolina ovat monen eri abstraktiotason aiheuttamat kiinteät kustannukset muistissa ja prosessointikapasiteetissa sekä kirjastoista ja järjestelmistä aiheutuvat kompleksiset riippuvuudet.

3.2 Keskusyksikkö vai näytönohjain

Näytönohjainten suurin etu on laitteiston ja toiminnollisuuksien erikoistuminen. Erikoistumisella tarkoitetaan keskittymistä vain tietokonegrafiikan informaation ja numeraalisten arvojen käsittelyyn. Nykyisin voidaan konservatiivisesti sanoa näytönohjainten suoritustehojen olevan moninkertaiset (2-10x) keskusyksikköön verrattuna kyseisissä operaatioissa. Tietorakenteiden luominen, käsittely ja yleinen hallinta voi olla erittäin hankalaa, kun toimitaan puhtaasti näytönohjainrajapinnan liukuhihnan mukaisesti. Esimerkiksi abstraktien rakenteiden kuten puiden tai verkkojen hyötykäyttö on huomattavasti haasteellisempaa näytönohjainten rajoitteellisemmassa ympäristössä. (10; 11; 12.)

Näytönohjainta vasten ohjelmoidessa muita rajoittavia tekijöitä ovat rajapinnat, tietovirtaväylät, muisti ja laitteisto. Rajapintoihin kuuluvat tiedonkuljetukseen liittyvät kiinteät kustannukset (engl. Overhead), väylien skaalaamattomuus suhteessa prosessointitehojen kasvuun (Mooren laki), muistin suhteellisen pieni määrä (vaikka onkin olemassa VRAM) ja valmistajariippuvaiset ominaisuudet (NVIDIA CUDA, AMD Mantle) ovat konkreettisia esimerkkejä rajoitteista. (13; 14.)

Keskusyksikköä vasten ohjelmoidessa on täysin vapaat kädet suunnittelusta ja toteuttamisesta. Se mahdollistaa haastavien tietorakenteiden ja algoritmien toteuttamisen. Keskusyksikköä vasten ohjelmoidessa rajoittavia tekijöitä ovat kirjasto- tai alustariippuvaisuudet, sovelluksen kompleksisuus, sovelluksen prosessoinnin kilpailutilanne keskusyksikön liukuhihnalla ja keskusyksikön geneerisen toteutuksen aiheuttamat tiedonkuljetuksen pysyvät tiedot. (14.)

3.3 Näytönohjainrajapinnat

Grafiikkamoottoria suunnitellessa joudutaan ottamaan kantaa siihen, pystytäänkö renderöimiseen tarvittavia laskutoimituksia kiihdyttämään näytönohjainten avulla sekä pystytäänkö näytönohjaimen renderöimä kuva siirtämään suoraan näytön kuvapuskuriin. Näytönohjainrajapinnat mahdollistavat kyseiset operaatiot käyttöjärjestelmää vasten ohjelmoidessa. Näytönohjainrajapintojen liukuhihnat voidaan toteuttaa sarjassa (OpenGL, Direct3D) tai rinnakkaisesti (CUDA, OpenCL), jolloin piirto-operaatiot skaalautuvat joko suoraan ytimen maksimilaskentakapasiteetin mukaan tai ydinten määrän mukaan. (13; 15; 16.)

3.3.1 OpenGL

OpenGL on alustariippumaton ohjelmointirajapinta (API) näytönohjainta vasten. Sen avulla voidaan renderöidä 2d- ja 3d-vektorigrafiikkaa näytönohjaimella kiihdyttäen. Teknologian kehittämisen aloitti vuonna 1991 Silicon Graphics, inc. Nykyisin sitä ylläpitää ja kehittää tuottoa tavoittelematon yritysten yhteenliittymä Khronos Group. (15; 17.)

OpenGL on jatkuvasti kehittyvä ohjelmointirajapinta ja se tukee monen eri ohjelmointikielen yhdistämismahdollisuutta. Siitä on lisäksi kehitetty sulautettujen järjestelmien versio OpenGL ES ja nettiselainten renderöinnin laitteistokiihdyttämistä varten WebGL. (17.)

OpenGL:n on suunniteltu asiakas- ja palvelinarkkitehtuurin mukaisesti tilakoneeksi. Tilallisten funktiokutsujen ansiosta OpenGL on pysynyt taaksepäin versioyhteensopivana ja mahdollisten tiedonkuljetukseen liittyvien pysyvien tietojen määrä on pienempi verrattuna esimerkiksi Direct3D:n olioiden lähettämiseen funktiokutsuissa. (17; 16.)

Ohjelmoitavana rajapintana OpenGL tarjoaa paljon erilaisia vakiotoimintoja, mutta se myös tukee laajennuksia, jotta kolmannen osapuolen kehittäjät pystyvät lisäämään toiminnallisuutta. Laajennuksia käytettäessä pitää muistaa, että alustariippumattomuutta ei voida taata. Esimerkiksi NVIDIA:n laajennusta käytettäessä AMD:n näytönohjaimet eivät osaa tulkita kyseisiä komentoja. (17; 10.)

OpenGL:n yksittäinen konteksti prosessoi näytönohjaimeen lähetetyt komennot sarjassa. Siitä johtuen erilaiset rinnakkaiset tekniikat ovat todella haastavia toteuttaa nykyisellä arkkitehtuurilla. (15.)

Alun perin OpenGL tuki ainoastaan ennalta määriteltyjä verteksien ja pikseleiden manipulaatio-operaatioita. Versiosta 2.0 lähtien on ollut mahdollista myös asiakasohjelman määrittää erilaisia sävytinohjelmia OpenGL:n liukuhihnalle. Sävytinohjelmat toteutetaan GLSL (OpenGL Shading Language) -sävytinkielen avulla, joka muistuttaa hieman C-ohjelmointikieltä tietorakenteiltaan ja syntaktisesti. Sävytinohjelmat ovat käytännössä vain komentosuoritteita sisältäviä tekstipätkiä, jotka lähetetään näytönohjaimelle käännettäväksi osaksi liukuhihnan prosessia. (15.)

3.3.2 Direct3D

Direct3D on Microsoftin kehittämä alustariippuvainen (Windows) matalan tason ohjelmointirajapinta (API) näytönohjainta vasten. Se on kehitetty kaksi- ja kolmiulotteisen vektorigrafiikan renderöimiseen. Se toimii OpenGL:n tavoin asynkronisesti asiakas- ja palvelinarkkitehtuurin mukaisesti. (16; 18.)

Direct3D koostuu monesta erilaisesta ohjelmointirajapinnasta. Esimerkiksi Windows Store app -ohjelman luomista varten ohjelmoija voi tarvita Direct3D 11-, DXGI- ja HLSL- rajapintoja. (16; 18.)

Direct3D:n rajapinnan versiot ovat laitteistoriippuvaisuuksien lisäksi myös käyttöjärjestelmän versiosta riippuvaisia. Esimerkiksi Direct3D 10 -versio vaatii vähintään Windows 7 -version ja Direct3D 9 vähintään Windows XP -version toimiakseen. (16; 18.)

HLSL(High-Level Shading Language) on korkean tason sävytinohjelmakieli, jolla voidaan OpenGL:n GLSL-sävytinkielen tapaan vaikuttaa näytönohjaimen liukuhihnan toimintaan. Se muistuttaa myös C-kieltä syntaktisesti. (16; 18; 19.)

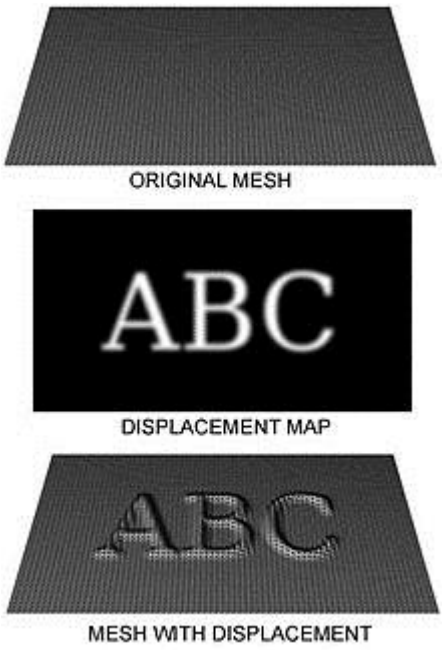
3.3.3 Yksinkertaistettu OpenGL ja Direct3D -liukuhihna

Kuviossa 3. mallinnetaan yksinkertaistettu malli OpenGL:n ja Direct3D:n abstrahoimasta liukuhihnasta. Liukuhihnaprosessi aloittaa lataamalla verteksi-, pikseli- ja tekstuuridatan näytönohjaimen muistiin.

Seuraavaksi näytönohjaimen muistissa dataan kohdistuu verteksikohtaisia operaatioita. Operaatioita on mahdollista manipuloida näytönohjainrajapinnan kautta näytönohjaimen muistiin ladattavan verteksisävytinohjelman (engl. vertex shader) avulla.

Verteksisävytinohjelma voidaan nimensä mukaisesti toteuttaa verteikseihin kohdistuvia operaatioita näytönohjaimella. Sävytinohjelma soveltuu geometrioiden manipuloimiseen, esimerkiksi malli-informaation liikutteluun, jossa verteksien sijaintivektorit transformoidaan spatiaaliseen avaruuteen projektio, näkymä ja mallimatriisien avulla. Se tarjoaa mahdollisuuden verteksien tekstuurikoordinaattien määrittelemiseen, valaistuksen mallintamiseen verteksien normaali- ja valonlähdesijaintivektorien avulla ja verteksikohtaisten tekstuurimateriaalien luomiseen. Operaatioiden rajapinnan määrittävät sävytinkielet (GLSL,HLSL) itsessään, joissa kerrotaan. kuinka erilaisia muuttujia ja tietorakenteita tulee kutsua.

Ennen primitiivien luomisprosessia liukuhihnalla on mahdollisuus tesseloimiseen. Tesseloidessa verteksi-informaatio pilkotaan helpommin käsiteltävään ja renderöitävään muotoon. Tesselointia voidaan soveltaa esimerkiksi siirtymäkuvaus (engl. displacement mapping) tekniikkaan. Siinä käytetään hyväksi tekstuurikarttaa, johon on tallennettu informaatio verteksien normaalin suuntainen syvyysvektori. Tesseloidessa syvyysvektorin avulla luodaan vertekseille uudet sijainnit normaalin suhteen. Kuviossa 2. näkyy yksinkertainen toteutus tekniikalle. (20; 21.)

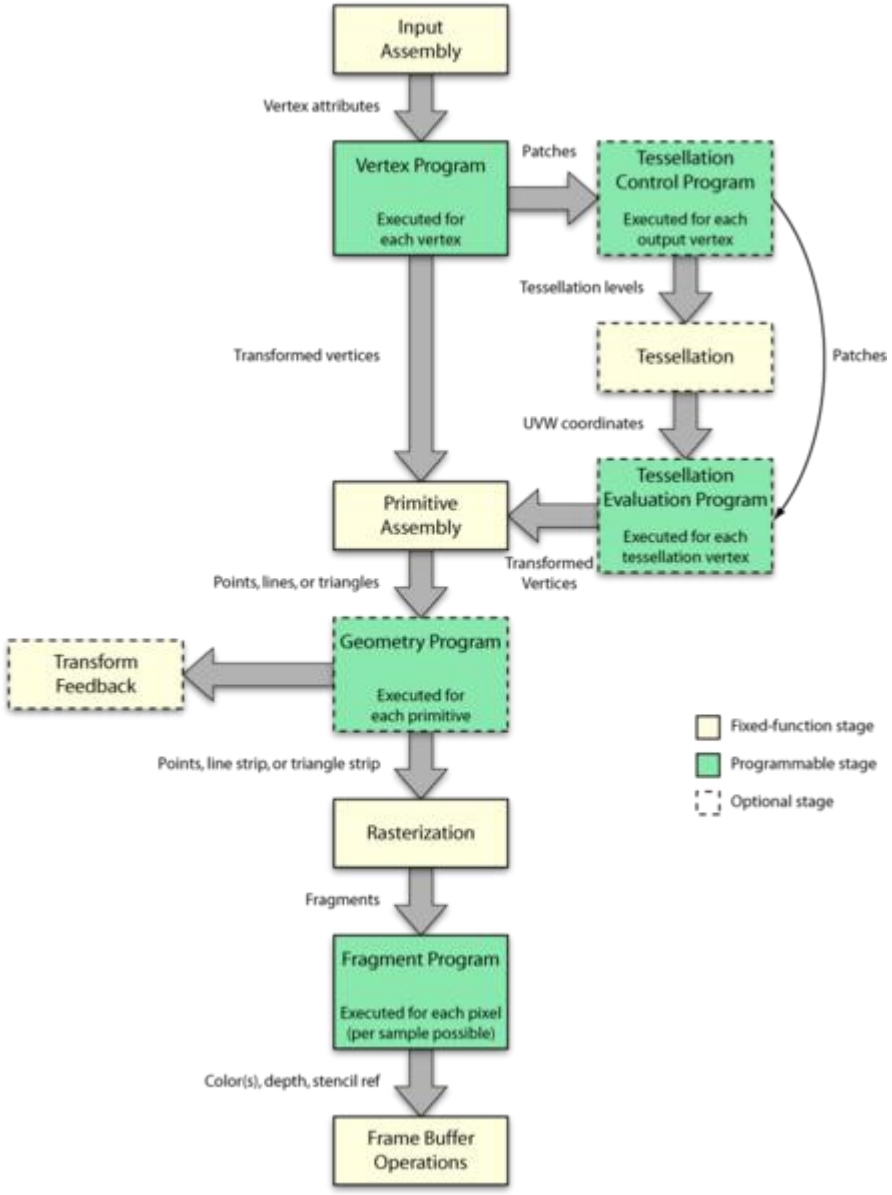


Kuvio 2. Siirtymäkuvausta sovelletaan yksinkertaiseen tasaiseen pintaan (71.)

Koostamisvaiheessa vertekseistä muodostetaan matemaattisia primitiivejä (kolmioita, monikulmioita tai pisteitä). Rajapinnan käyttäjällä on mahdollisuus vaikuttaa tähän vaiheeseen ohjelmallisesti geometriasävytinohjelman avulla. Geometriasävytinohjelmaa voidaan soveltaa manipuloimaan verteksien yhdistävien viivojen paksuuteen tai tarkkuuteen, jolloin voidaan saavuttaa partikkelijärjestelmän tarvitsemia ominaisuuksia.

Rasterointivaiheessa tekstuurit ja primitiivit yhdistetään fragmenteiksi kaksiulotteiseen kuvaan. Kuvan pisteisiin on tallennettu primitiivi-, väri- ja mahdollisesti syvyysarvo. Kuvaa on mahdollista manipuloida rajapinnan käyttäjänä fragmenttisävytinohjelman avulla. Sävytinohjelmassa voidaan soveltaa erilaisia tekniikoita fragmentin värin määrittämiseksi syvyysarvojen tai muiden näytönohjaimen tarjoamien puskureiden avulla. (22; 23.)

Lopulta muodostetut fragmentit viedään näytönohjaimen kuvapuskuriiin, josta pikselit kartoitetaan resoluution mukaan muistiin tai suoraan näyttölaitteille.



Kuvio 3. Yksinkertainen mallinnus OpenGL- ja Direct3D-liukuhihnasta (72.)

3.3.4 OpenCL

OpenCL on Khronos Groupin kehittämä alustariippumattomaan rinnakkaisohjelmointiin tarkoitettu sovellusrunko, jossa alustat voivat olla näytönohjaimia, keskusyksiköitä, digitaalisia signaaliprosessoreita, digitaalisia mikropiirejä ja muita prosessoreita. (24; 25.)

OpenCL voidaan soveltaa esimerkiksi tähtitieteeseen, biologiaan, kemiaan, fysiikkaan, tiedonlouhintaan, taloustieteisiin ja muihin laskentaa vaativiin aloihin. (24; 25.)

OpenCL muodostaa laskentajärjestelmän monesta eri laskentalaitteesta, jotka voivat olla keskusyksikköjä tai näytönohjaimia sekä isäntäprosessista (keskusyksikkö). OpenCL tarjoaa tavan luoda C-tyylisiä ohjelmia (termi kernel), joita pystytän ajamaan rinnakkaisesti laskentalaitteilla. Kernel-ohjelmia ja laitteiston muistia käskytetään ohjelmoitavan rajapinnan kautta isäntäprosessista. (24; 25.)

OpenCL:n ohjelmat käännetään ajettaessa, jotta alustariippumattomuus pystytään takamaan. Kehittäjä pääsee kirjastojen ja Khronos Groupin kehittämien kääntäjien ja laajennuksien avulla OpenCL:n toiminnollisuuksiin käsiksi. Kääntäjät kääntävät ohjelman suoraan C- ja C++-kielistä. Laajennuksien kautta toiminnallisuudet saadaan myös Python-, Julia- ja Java-kielille. (24; 25.)

OpenCL:n muistiavaruus jakaantuu neljään eri tasoon. Globaali taso on kaikkien prosessien jakama muistialue. Se sisältää paljon käsittelyviivettä. Lukuoikeudellinen taso on pienempi ja pienen käsittelyviiveen omaava muistialue. Siihen pystyy ainoastaan isäntäprosessi kirjoittamaan. Paikallinen taso on laskentalaitteiden laskuprosessien sisäisesti jaettu muistialue. Prosessitasoinen muistialue toimii ainoastaan prosessille rekisterinä. (24; 25.)

3.3.5 NVIDIA CUDA

CUDA(Compute Unified Device Architecture) on NVIDIA:n kehittämä rinnakkaisohjelmoinnin alusta, jolla voidaan valjastaa näytönohjaimen prosessit rinnakkain ajettaviksi laskentatoiminnoiksi. (13; 26.)

CUDA mahdollistaa kehittäjälle suoran pääsyn näytönohjaimella prosessoitaviin elementteihin virtuaaliseen käskykannan ja muistin avulla. CUDA tukee ainoastaan NVIDIA:n näytönohjaimia prosessointialustana. (13; 26.)

CUDA:n avulla näytönohjainta voidaan soveltaa yleisesti laskentaan tai prosessointiin eikä ainoastaan tietokonegrafiikkaan. Sovelluskohteet ovat hyvin samantapaisia kuin OpenCL:ssä. (13; 26.)

Kehittäjät pääsevät CUDA alustaan käsiksi CUDA kiihdytettyjen kirjastojen, kääntäjä direktiiveillä sekä NVIDIA:n tuottamilla laajennetuilla kääntäjillä, jotka tukevat C, C++ ja

Fortran kieliä. CUDA tukee myös muita laskennallisia rajapintoja kuten OpenCL, Microsoftin DirectCompute ja C++ AMP. (13; 26.)

4 Realistisuus ja reaaliaikaisuus

Tietokonegrafiikkaa luodessa tulee vastaan seuraavanlainen kysymys, kuinka saada kuvasta mahdollisimman aidon näköinen suhteessa subjektiivisiin kokemuksiin havaitsemastamme maailmasta. Digitaalisessa maailmassa vastaus on hyvin usein, että tarvitaan lisää pikseleitä ja pikselin määrittäviä ominaisuuksia. Jos kysymystä tarkastellaan syvemmin päästään vastauksiin, joissa pitää ottaa kantaa laitteiston laskentakapasiteettiin ja kuvan realistisuuteen.

Realistisuudella tässä kontekstissa tarkoitetaan subjektiivista kykyä erottaa fotoneista koostetetun digitaalinen kuvan luomaa kokemusta suhteessa täysin tietokoneella generoituun kuvaan luomaan kokemukseen. Reaaliaikaisella tarkoitetaan kuvien renderöimisen välisen viiveen minimoimiseen, jolla voidaan luoda illuusio liikkuvasta kuvasta.

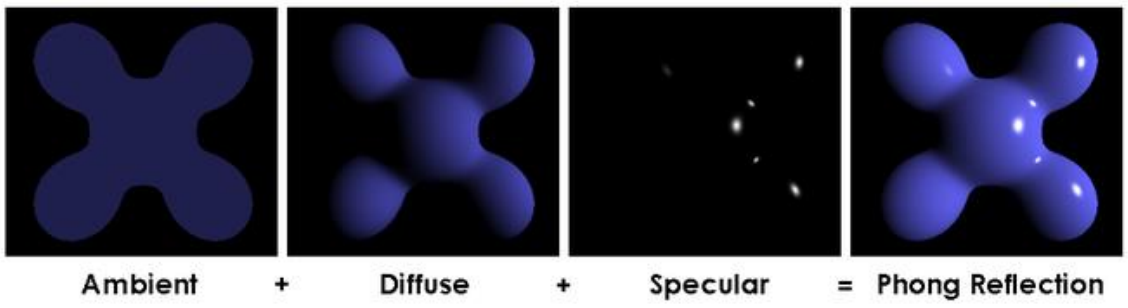
Tehokkaan ja reaaliaikaisen ohjelmistosovelluksen visuaalisesti realistiseen toteuttamiseen joudutaan ratkomaan laskentatehokkuuden ongelmaa. Ongelma voidaan ratkaista käyttämällä algoritmeja, jotka pyrkivät karkeasti imitoimaan ja approksimoimaan oikean fysikaalisen maailman visuaalisuutta välittämättä tarkemmin fysiikan teorioista. Tällaisissa algoritmeissa painotusarvona on se, että kuva näyttää luonnolliselta silmälle verrattuna siihen, että ne ovat matemaattisesti todistettavissa totuusarvoisiksi fysikaalisten mallien perusteella. Seuraavana on lyhyt kooste yleisimmistä tekniikoista, joita käytetään tietokonegrafiikan parissa tehokkaina approksimoivina algoritmeina.

4.1 Laskutehokkaat tekniikat

4.1.1 Phongin sävytys- ja heijastusmallit

Bui Tuong Phong kehittämä ja vuonna 1973 väitöskirjassaan julkaisema approksimoiva sävytysmalli interpoloi geometrioiden verteksien normaalien avulla geometrioiden pintojen pikseleiden värin. Sävytysmallin pohjalle perustuu myös Phong-heijastusmalli, joka approksimoi lokaalin valaistuksen heijastumista. Heijastusmalli koostuu kolmesta

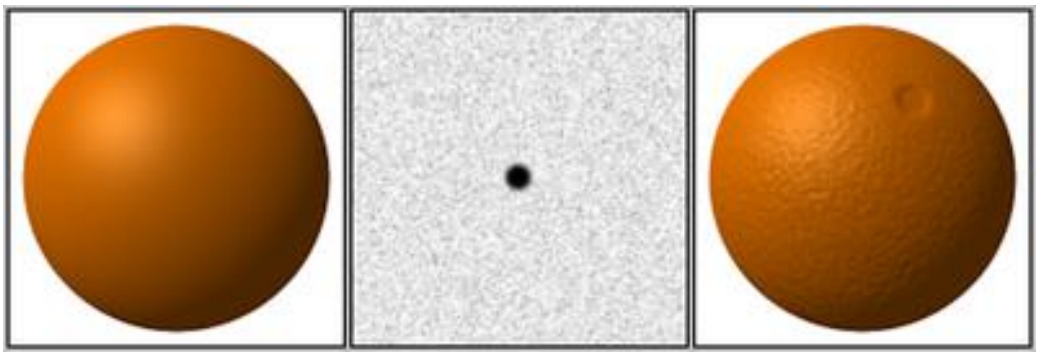
eri komponentista: taustavalosta, mattaheijastuksesta ja peiliheijastuksesta. Se toimii realistisimpana vaihtoehtona muovisten pintojen jäljitelmissä, kuten kuviossa 4 voidaan hyvin se havannoida. (5; 27; 28.)



Kuvio 4. Phongin sävytys- ja heijastumallin eri komponentit (61.)

4.1.2 Kohoumakuvaus

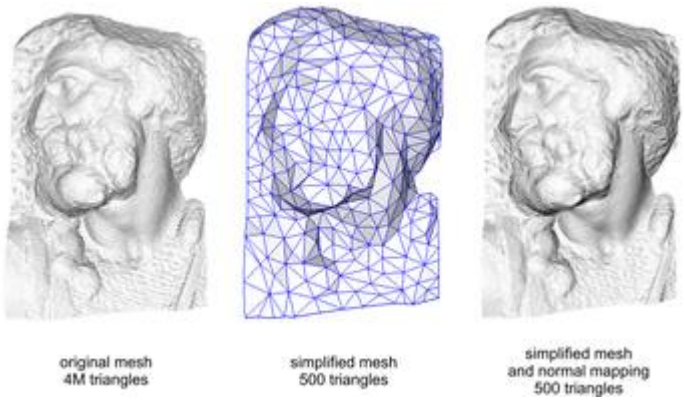
Kohoumakuvauksen luojana pidetään James Blinniä vuonna 1978. Tekniikka perustuu geometrioiden pintojen verteksien normaalien siirtämiseen kohoumakuvaustekstuurin avulla. Tekstuuriin on tallennettu arvo normaalin siirtoa varten. Kuten alla olevassa kartassa näkyy, pikselin tummuus on suhteellinen normaalin siirrettävän arvon määrään. Kun pintojen muutettuja normaaleja käytetään valaistus- tai sävytysmalleissa luodaan kuva siitä, että pinta olisi rosoinen tai epätasainen (esimerkiksi Phongin sävytysmallissa). Tekniikka ei kuitenkaan vaikuta geometrian vertekseihin, joten läheltä ja tietyistä valaistuskulmista nähdään geometrian "tasaisuus". Kuviossa 5. demonstroidaan tekniikkaa. (5; 27; 29.)



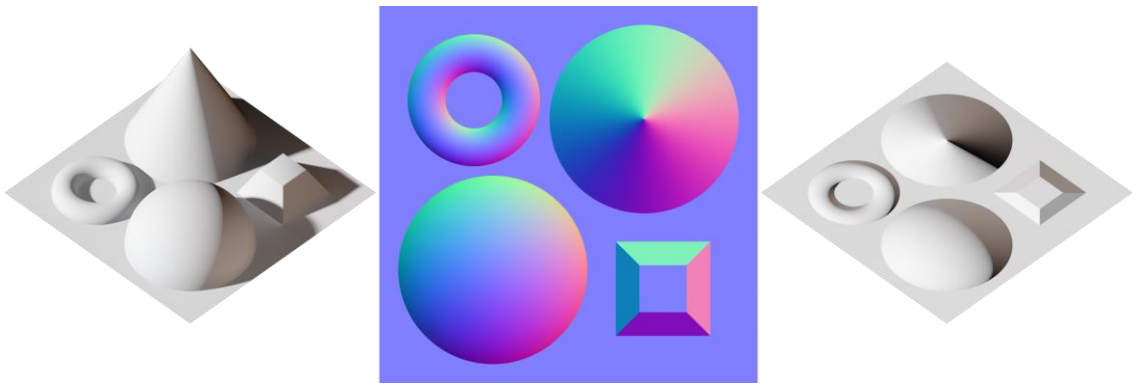
Kuvio 5. Kohoumakuvaus sovellettuna tasapintaiselle pallolle (62.)

4.1.3 Normaalikuvaus

Normaalikuvaus on kohoumakuvauksen kehittyneempi toteutus. Kohoumakuvauskarttaan tallennetaan syvyyden lisäksi kaksi muuta ulottuvuutta, josta muodostetaan vektori. Vektoria käytetään valaistusmallin laskutoimituksissa, jolloin saavutetaan yksityiskohdattomien mallien visuaalisuuden parannus lisäämättä käsiteltävien verteksien määrää. Kuvioissa 6. ja 7. demonstroidaan tekniikan vaikutuksia. (30.)



Kuvio 6. Vertailu eri tekniikoilla luoduista kuvista (63.)

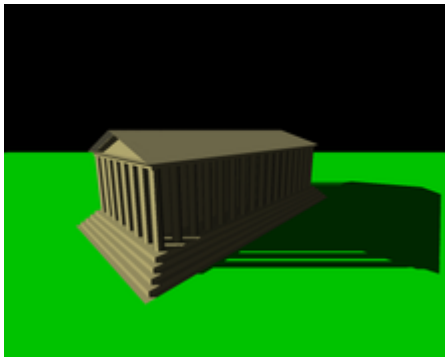


Kuvio 7. Normaalikuvauksen tekstuuri ja sen vaikutus (64.)

4.1.4 Varjokuvaus

Varjokuvaus on tekniikka, jolla luotuun kuvaan luodaan valaistuksesta riippuvaisia varjostuksia näkymässä oleville geometrioille. Algoritmi toimii kaksivaiheisesti. Ensimmäisessä vaiheessa valaistuksen kautta projisoidaan näkymä, jossa lasketaan näkymän fragmenteille syvyysarvo valaistuksen suhteen. Syvyysarvo tallennetaan tekstuuriin,

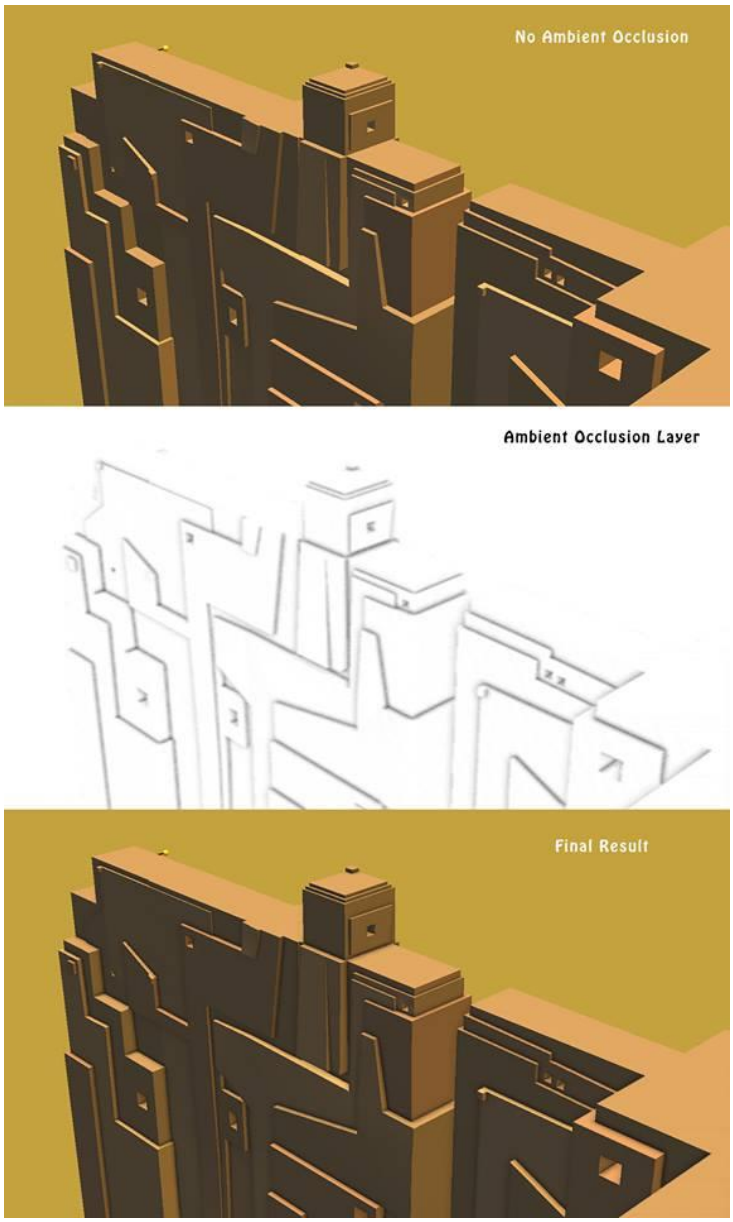
jota käytetään seuraavassa vaiheessa hyväksi. Toisessa vaiheessa näkymä renderöidään normaalisti, mutta fragmenttien väritysvaiheessa tekstuurin syvyysarvoja verrataan sen hetkisen fragmentin valaistuksesta projisoituun syvyysarvoon. Jos projisoitu syvyysarvo on pienempi kuin syvyystekstuurin, niin kyseinen fragmentti ei ole varjon peitossa. Kuviossa 8. on esimerkki tekniikan vaikutuksista. (31.)



Kuvio 8. Näkymä, jossa on toteutettu varjokuvaus (65.)

4.1.5 Taustavalaistuksen okkluusio

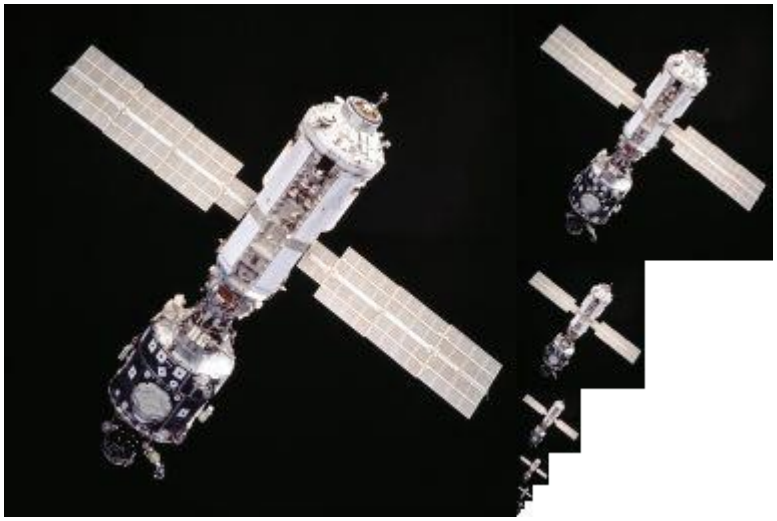
Tekniikka pyrkii simuloimaan taustavalaistuksen vaikutusta näkymän näkyviin pisteisiin. Mitä tummempi alue, sitä vähemmän pisteeseen heijastuu valoa. Esimerkiksi simuloidessa kirkkaan taivaan valaistusta voidaan valaistulle pisteelle suoraan laskea suhteellinen näkyvä taivas, jonka avulla pisteen tummuus lasketaan. Toisin kuin lokaali Phongin sävytysmalli on taustavalaistuksen okkluusio approksimoiva globaali valaistusmalli. Tekniikka soveltuu parhaiten juuri kirkkaan keskipäivän auringon mallintamiseen tai sisätilojen valaistukseen, kun valaistus on suoraan kohden pistettä. Kuviossa 9. näytetään tekniikan eri vaiheiden vaikutukset näkymään. (32.)



Kuvio 9. Taustavalaistuksen okkluusion vaikutus näkymään (66,)

4.1.6 MIP-kuvaus

Lyhenne MIP tulee latinan sanoista multum in parvo, joka tarkoittaa "paljon pienessä". MIP-kuvat sisältävät optimoituja esirenderöityjä tekstuureja, jotka skaalautuvat tarvittavan resoluution mukaan. Tekstuurin korkeus ja leveys pienenee neliöjuuren verran edelliseen tasoon nähden. Korkearesoluutioisia kuvioita käytetään lähellä olevissa kappaleissa ja matalan resoluution kuvia käytetään kaukana olevissa kappaleissa. Kuviossa 10. esitetään MIP-kuvaa. (5; 27; 33.)



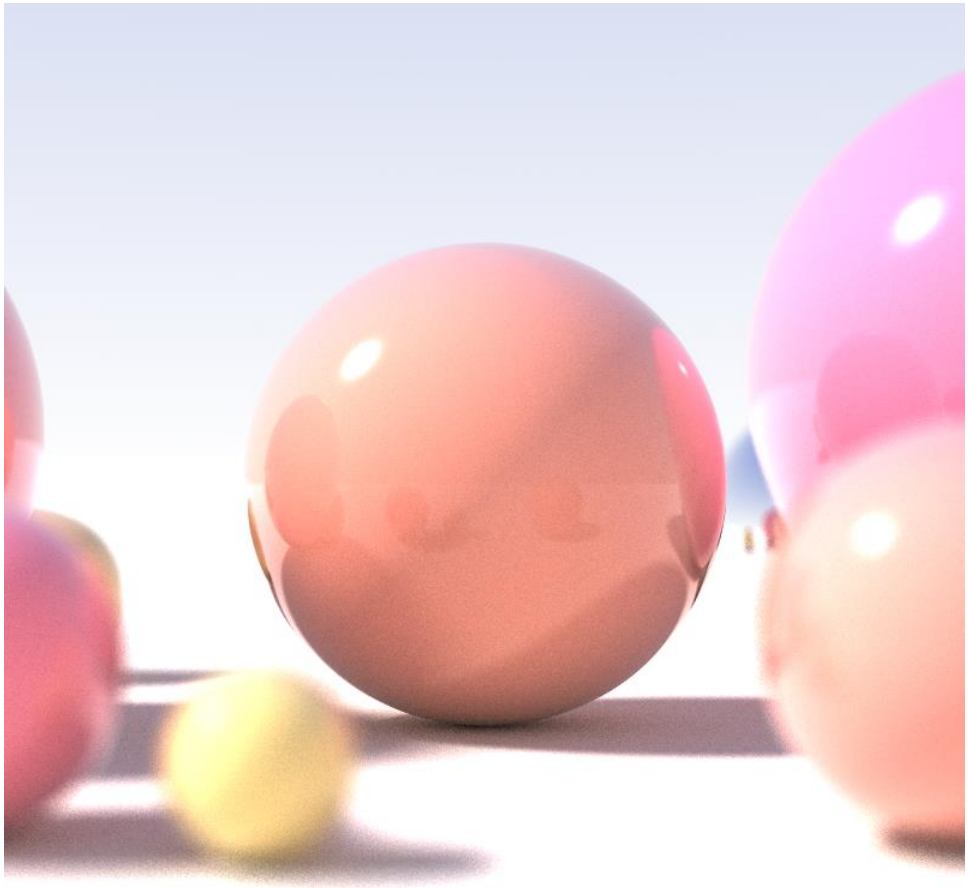
Kuvio 10. MIP-tekstuuri (67.)

4.2 Fotorealistiset tekniikat

Kun kuvan luomisprosessi ei ole aikarajoitteinen, voidaan tietokonegrafiikan tekniikoilla saavuttaa parhaimmillaan fotorealistinen laatu. Algoritmien pohjana käytetään fysikaalisia malleja tai matemaattisia yhtälöitä.

4.2.1 Säteenjäljitys

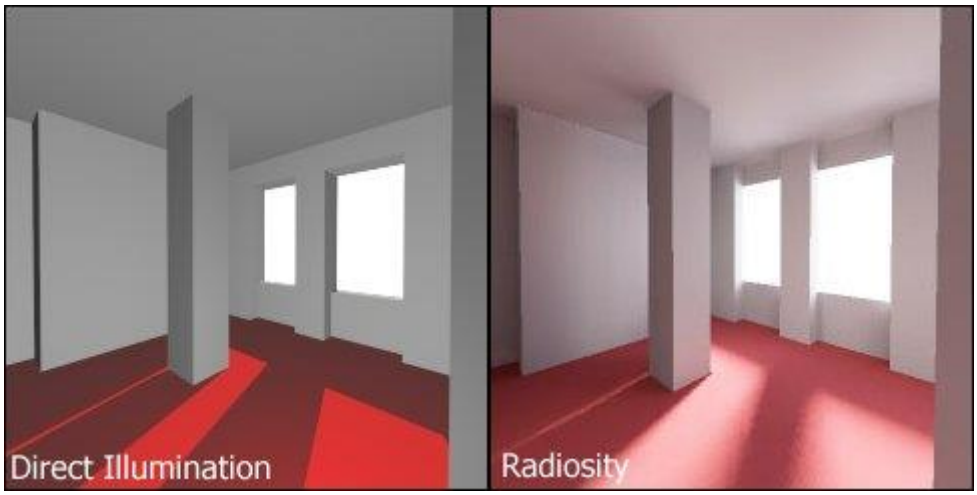
Globaali valaistusmalli perustuu simuloituihin valonsäteisiin, jotka kuljettavat kameranäkymän kautta nähdyn kuvan pikselien määrittävää väri-informaatiota. Kamerasta lähtevät säteet törmäytetään osalla näkymän kappaleista, ja törmäyskohdassa algoritmi tarkastelee valaistuksen ja objektin pinnan materiaalien yhteisvaikutusta, josta syntyy pikselin lopullinen väri. Algoritmi sisältää monta laskutoimitusta per pikseli ja sen vuoksi skaalautuu erittäin huonosti sarjallisissa operaatioissa. Säteenjäljitys on erittäin hyvä kandidaatti rinnakkaisesti toteutettavaan renderöintiliukuhihnaan, koska algoritmin operaatiot ovat suurimmaksi osin atomisesti suoritettavia. Kuvio 11. edustaa kuvaa, jossa tekniikka on sovellettu. (5; 27; 34.)



Kuvio 11. Säteenjäljityksellä tuotettu kuva (68.)

4.2.2 Radiositeetti

Radiositeettimenetelmä on globaali valaistusmalli, jonka algoritmi perustuu lämpösäteilyn mallintamiseen. Algoritmi pyrkii simuloimaan suoran valaistuksen aiheuttamaa mattaheijastumista muille pinnoille. Algoritmia voidaan soveltaa tehokkaaseen renderöimiseen, kun näkymän pinnat ovat esirenderöity tekstuureiksi. Ilman esirenderöityjä tekstuureita tai muita optimointeja laskutoimituksien määrä kasvaa toisen asteen yhtälön mukaisesti, kun pintojen ja geometrioiden määrä kasvaa lineaarisesti. Radiositeetin vaikutusta voidaan hyvin havainnoida kuviosta 12. (5; 27; 35.)



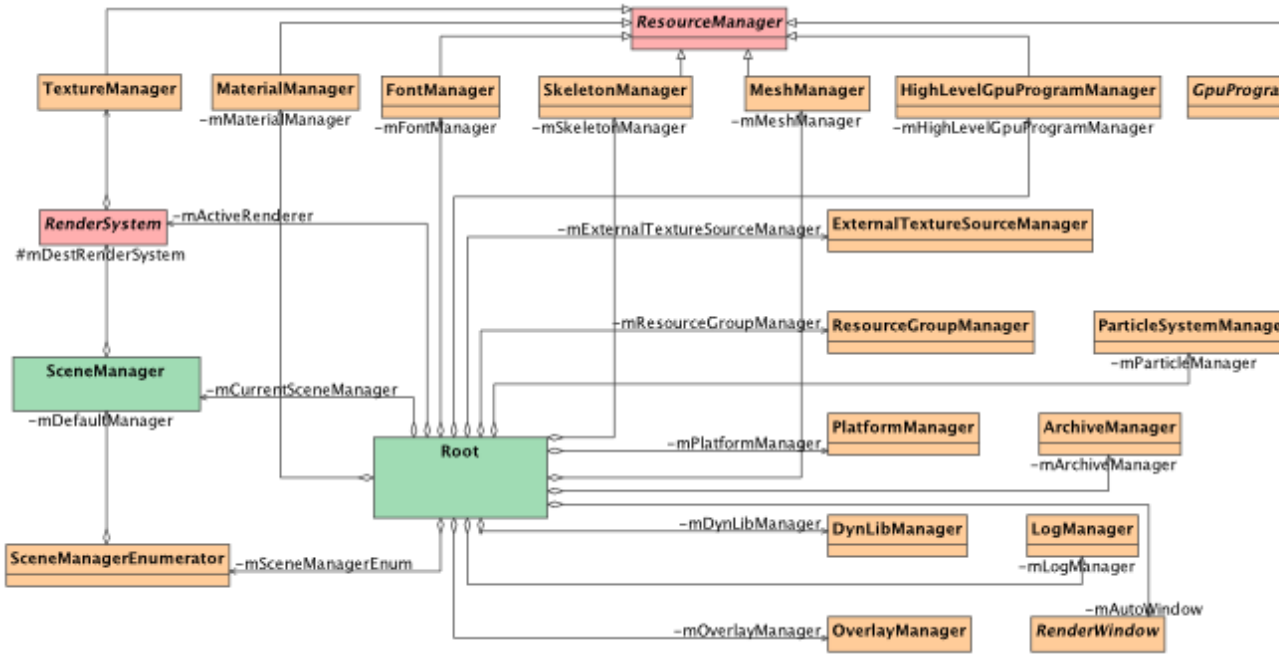
Kuvio 12. Suoran valaistuksen ja radiositeettimenetelmän luoman kuvan ero (69.)

5 Toteutuksia

5.1 OGRE

OGRE (Object-oriented Graphics Rendering Engine) on avoimen lähdekoodin tietokonegrafiikan renderöintisovellus, jota ylläpitää ja kehittää pieni tiimi (The OGRE Team) kasvavan yhteisön avuin. OGRE tarjoaa laajan oliopohjaisen rajapinnan kolmiulotteisten näkymien renderöimiseen ottamatta kantaa alemman tason toteuttajiin tai rajapintoihin. OGRE on selkeästi ja järjestelmällisesti suunniteltu ja dokumentoitu kaikkien luokkien osalta. OGRE tarjoaa tuen Direct3D:n (DirectX 9 ja 10) ja OpenGL:n näytönohjainrajapinnoille: Windows-, Linux- ja MAC OS X -käyttöjärjestelmille. Koodi voidaan kääntää Visual C++ 2003-, 2005-, 2008-, 2010- ja gcc 4+ -kääntäjillä. (36; 37.)

Kuvio 13 on pelkistetty otos OGRE:n arkkitehtuurista. Siinä nähdään Root-olio, joka järjestää ja hallinnoi kaikkia ylempien tasojen olioita. Näihin kuuluu renderöintijärjestelmät, näkyvyyden hallinta, resurssien hallinta, ikkunointi ja erilaisten lisäosien lataaminen. Root-olion avulla konfiguroidaan ja alustetaan järjestelmä. Suurin osa OGRE:n luokista voidaan jakaa kolmeen pääryhmään renderöintiin, näkyvyyden ja resurssien hallintaan. OGRE:n modulaarisuus tekee siitä erittäin helposti integroitavan ja laajennettavan. Kuviossa 14. on OGRE:n tuottama kuva.(36; 37.)



Kuvio 13. Yksinkertaistettu malli OGRE:n arkkitehtuurista (70.)

OGRE:n päätoiminnallisuuksiin kuuluu:

- tukee erilaisia materiaaleja ja sävytinohjelmia
- sisältää merkintäkielen, jolla voidaan määritellä ja hallita materiaalisia resursseja koodin ulkopuolella
- tukee verteksi- ja fragmenttisävytinohjelmia, kummatkin ohjelmat voidaan toteuttaa alemman tason ohjelmana :assembler kielellä ja korkeamman tason ohjelmana: Cg(NVIDIA:n sävytinkieli), Direct X 9 HLSL (Direct3D sävytinkieli) tai OpenGL GLSL avulla
- OGRE tarjoaa automaattisen tuen monille yleisille pysyville parametreille kuten maailmaperspektiivin matriisi, valaistustilojen informaatio tai mallitason kuvakulman positio jne.
- tarjoaa suuren valikoiman kiinteitä funktio-operaatiota kuten moninkertaista teksturointia, moniläpiajoisia toiminnollisuuksia, tekstuurikoordinaattien generointia ja muokkausta, atomisia väritysoperaatioita ja alfakanavan manipulointioperaatioita
- tukee monenlaisia materiaalitekniikoita eri laitteille, joista OGRE valitsee parhaiten optimoidut ja tuetuimmat
- tukee materiaalien LOD(Level of Detail) hallintaa; materiaalien kustannukset laskevat kun niitä käyttävä objekti loittonee niistä

- tukee tekstuurien lataamista PNG, JPEG, TGA, BMP, PVRTC tai DDS tiedostoformaateista, sisältäen myös harvinaisemmat 1D tekstuurit, volumetriset tekstuurit, kuutiokuvaukset, HDR-kuvat ja kompressoidut tekstuurit (DXT/S3TC)

- tukee dynaamisia tekstuureja kaikissa tuetuissa formaateissa, tehokkaaseen elokuvien tai reaaliaikaisen sisällön toistoon tekstuurin avulla

OGRE hallinnoi käyttämiään resursseja seuraavien toimintojen kautta:

- tukee joustavasti mallinnusverkko dataformaatteja.

- erottelee verteksien puskurit, indeksien puskurit, verteksien määrittelyt ja puskureiden kuvaukset

- tukee Milkshape3D, 3D Studio Max, Maya, Blender ja Wings3D mallinnustyökalujen formaatteja

- tukee luurankoanimaatioita, sisältäen monien animaatioiden sekoituksia ja painoarvoilla määriteltävän skinning-järjestelmän

- tukee Bezier-paikkauksia kaareville pinnoille

- tukee asteittain muuttuvaa mallinnusverkkoa (LOD)

- tukee OpenGL-tyylisiä primitiivien luomista

OGRE hallinnoi näkyvyyttä seuraavan laisten ominaisuuksien kautta:

- täysin konfiguroitava ja joustava näkyvyyden hallinta, joka ei ole sidonnainen vain yhteen näkyvyystyyppiin

- yleinen SceneManager-luokka, joka poimii näkyvät geometriat niiden ympäröivien rajaustilavuuksien perusteella

- Octree-lisäosa mahdollistaa yleisten näkyvyyksien hallinnan octree-puurakenteen avulla

Kuvio 14. OGRE:n generoima kuvasta (71.)

5.2 Horde3D

Horde3D on avoimen lähdekoodin alustariippumaton grafiikkamoottori, joka pyrkii samanlaisiin ominaisuuksiin kuten OGRE. Tavoitteena on kuitenkin pitää tekniikat ja toiminnallisuudet mahdollisimman kevyinä prosessoitaviksi. Se soveltuu erityisesti isojen ryhmien massasimulointiin. Grafiikkamoottorin kehittäjinä ovat toimineet itsenäinen kehittäjäryhmittymä pyropix ja Augsburgin yliopisto. Kuviossa 15. on Horde3D:n luoma kuva. Horde3D:n päätoiminnallisuuksia ovat: (38; 39.)

- moderni sävytinarkkitehtuuri, joka toimii Shader Model 2.0 tukevalla laitteistolla
- alustariippumaton yhteensopivuus OpenGL piirtorajapinnan vuoksi
- suunniteltu yleisesti hyvin kevyeksi ja mahdollisimman riippumattomaksi välttäen monimutkaisuutta missä vain mahdollista

- moottorin koodi on toteutettu olio-ohjelmoiden optimoidulla C++-koodilla.
- modulaarinen ja korkeasti abstrahoitu C-tyylisen DLL ohjelmointirajapinnan avulla
- C-tyylinen DLL API mahdollistaa Horde3D:n käytön melkein millä tahansa ohjelmointikielellä
- integroituu helposti pelimoottoreiden ja muiden kolmannen osapuolen sovelluksien kanssa

Horde3D hallinnoi näkyvyyttä seuraavan laisten ominaisuuksien kautta:

- sisältää automatisoidun sisäisen muistin roskienkeräyksen.
- rajapinta mallien datan lataamiseen tiedostoista, tietovirroista tai arkistosta
- nopeasti uudelleen ladattavat resurssit mahdollistavat nopean tuotantoprosessin

Kuvio 15. Horde3D:n generoima kuva (72.)

5.3 POV-Ray

POV-Ray (Persistence of Vision Raytracer) on alustariippumaton avoimen lähdekoodin säteenjäljitysohjelma. Alun perin perustui David Kirk Buckin ja Aaron A. Collinsin kehittämään ohjelmaan DKBTrace. Nykyisin POV-Ray:tä kehittäjänä toimii The POV-Team. Kuviossa 16. on POV-Ray:n tuottama kuva. (40; 41.)

POV-Rayn päätoiminnallisuudet voidaan listata seuraavanlaisesti:

- sisältää näkymäselitekielen, jolla käyttäjä määrittelee kuvan sisältämät objektit, tekstuurit ja muut lopputulokseen vaikuttavat parametrit
- näkymäselitekieli tukee makroja ja silmukoita.
- sisältää kirjastoja, joista löytää ennalta määriteltyjä näkymiä sisältäen kappaleita ja tekstuureita.
- tukee erilaisia matemaattisia primitiivejä ja rakennettavia geometrioita
- tukee monen tyyppisiä valonlähteitä
- tukee tunnelmallisia graafisia efektejä(sumu, savu ja pilvet)
- tukee valon heijastumista, taittumista, valon simulointia fotonien kuvauksella
- tukee proseduurillisia tekstuureja sekä kohoumakuvausta
- tukee valaistuksen radiositeettia
- tukee erilaisia tekstuuriformaatteja tiedostomuodoissa TGA, PNG, JPEG.

Kuvio 16. POV-Ray:n muodostama fotorealistinen kuva (73.)

6 Oma toteutus

6.1 Idea

Ajatuksena oli luoda dynaaminen alustariippumaton korkean abstraktiotason grafiikkamoottori. Dynaamisella tässä kontekstissa tarkoitetaan reaaliajassa tulkattavia koodeja ja resursseja.

Dynaamisuudella halutaan luoda mahdollisimman pieni prosessisykli koodin toteutuksen ja testauksen välille, jolloin ominaisuuksien prototyypittäminen ja toteuttaminen nopeutuu huomattavasti.

Alustariippumattomuudella tarkoitetaan sitä, että yhteinen koodikanta voidaan kääntää mille tahansa valtavirran käyttöjärjestelmälle (MAC, Windows, Linux), ja toiminnollisuudet pysyvät samoina käyttöjärjestelmästä riippumatta.

Projektin tarkoituksena oli toteuttaa sovellus, joka on mahdollisimman yksinkertainen, reaaliaikainen, oliopohjainen, modulaarinen ja skaalautuva grafiikkamoottori. Sovelluksen tulisi toimia myös kehitysrunkona tulevaisuuden kehitysprojekteille.

6.2 Projektin rajaus

Halusin rajata projektin selkeästi rajaamalla grafiikkamoottorin ominaisuudet ohjelmointirajapinnan kautta (API), koska sisäisten ominaisuuksien määrittäminen olisi ollut huomattavasti työläämpää ja vaikeammin havainnollistettavampaa ulkopuolisille. Seuraavat käyttäjätarinat olivat mielessä suunnitellessani projektia.

Kehittäjänä pystyn kommunikoimaan grafiikkamoottorin kanssa reaaliaikaisesti rajapinnan kautta ilman, että tarvitsee kääntää koodia.

- Kehittäjänä pystyn lataamaan geometrian, johon pystyn liittämään yksinkertaisen kolmiulotteisen mallin käyttöjärjestelmästä riippumatta grafiikkamoottorin käytettäväksi.
- Kehittäjänä pystyn lataamaan geometrian, johon pystyn liittämään yksinkertaisen tekstuurin käyttöjärjestelmästä riippumatta grafiikkamoottorin käytettäväksi.
- Kehittäjänä pystyn lataamaan geometrian, johon pystyn liittämään vertekisävytinohjelman käyttöjärjestelmästä riippumatta grafiikkamoottorin käytettäväksi.
- Kehittäjänä pystyn lataamaan geometrian, johon pystyn liittämään fragmenttisävytinohjelman käyttöjärjestelmästä riippumatta grafiikkamoottorin käytettäväksi.
- Kehittäjänä pystyn manipuloimaan geometriaa skaalaamalla, kiertämällä ja siirtämällä sitä.
- Kehittäjänä pystyn manipuloimaan geometriaa skaalaamalla, kiertämällä ja siirtämällä reaaliaikaisesti.

- Kehittäjänä pystyn asettamaan yksinkertaisen valaistuksen geometrialle reaaliaikaisesti.
- Kehittäjänä pystyn asettamaan yksinkertaisia materiaaleja geometrialle reaaliaikaisesti sävytinohjelman avulla.
- Kehittäjänä saan itselleni näkyviin ikkunoinnin, jossa näen reaaliaikaisesti grafiikkamoottorin generoimat kuvat.
- Kehittäjänä pystyn vaikuttamaan kuvan projektioon ja perspektiiviin näppäimistön tietovirtojen avulla ja ohjelmallisesti.

6.3 Tutkimusvaihe

Tutkimisvaiheeseen kuului mahdollisimman monen erilaisen grafiikka- ja pelimoottoreiden ja näytönohjainten rajapintojen ominaisuuksiin tutustuminen, nettikurssien läpikäymistä, aiheeseen liittyvän kirjallisuuden opiskelemista, tietokonegrafiikkaan liittyvään matematiikkaan perehtymistä ja suurimpien tietokonegrafiikkateknologiajättien uutisvirtojen, dokumenttien ja foorumeiden seurantaa. Tiedon hankinta ei loppunut pelkästään tutkimusvaiheeseen, vaan jatkui iteratiivisesti projektin edetessä.

6.4 Projektin hallinta

6.4.1 Projektin kehitysympäristö

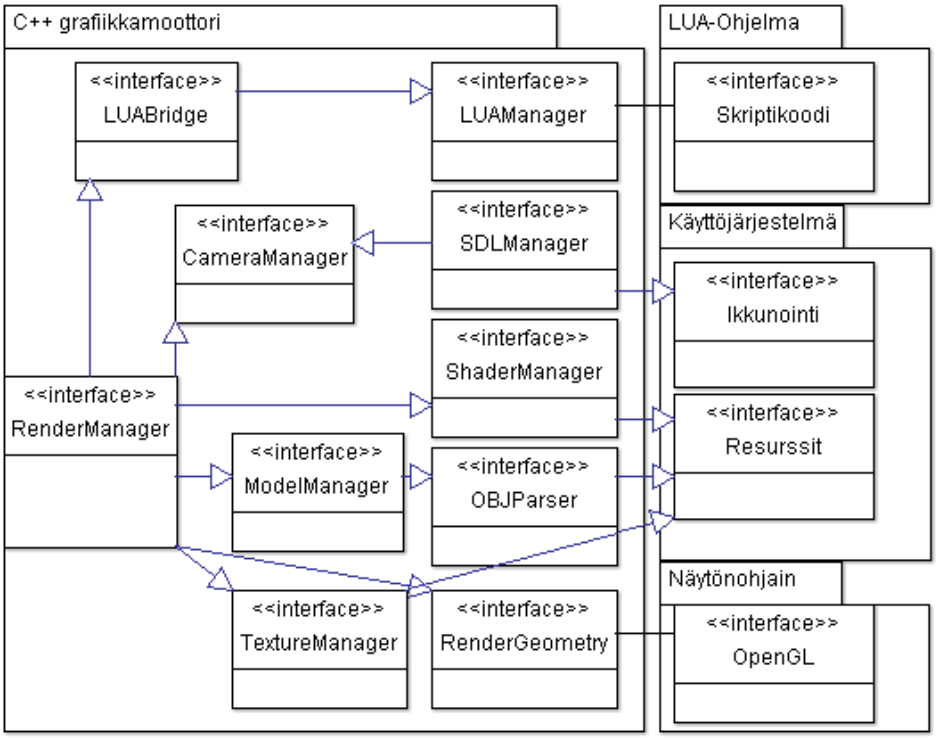
Kehitysprosessi tapahtui Windows 7 -käyttöjärjestelmässä ja koodin hallintaan käytin Microsoftin Visual Studio 2010 -kehitysympäristöä. Sen tarjoamin ominaisuuksilla pystyin muokkaamaan ja hallinnoimaan koodia sekä kääntämään koodin oikeiden riippuvuuksien kanssa. (42.)

6.4.2 Projektin ylläpito ja seuranta

Koodin versionhallintaan käytin avoimen lähdekoodin sovellusta nimeltä GIT. Projektin etenemisen seuraamista varten tarvitsin muutaman tekstitiedoston, josta löytyivät käyttötapaukset, hyväksymiskriteerit ja tieto kehityksen tilasta. (43.)

6.5 Ohjelma-arkkitehtuurin komponenttien suunnittelu

Kuviossa 17 on mallinnettu karkeasti toteutuksen luokkien ja olioiden kommunikaatio muiden toteutukseen liittyvien komponenttien kanssa. Siinä havainnoidaan hyvin sovelluksen modulaarisuus sekä ulkoisten tietovirtojen merkitys sovellukselle.



Kuvio 17. Malli oman sovelluksen arkkitehtuurista

6.5.1 Ohjelmointikieli

Ohjelmointi kieleen vaihtoehtona olivat C++, Java ja Python. Javassa ja Pythonissa ongelmina olivat muun muassa sisäiset ajonaikaiset muistinhallintatoiminnat, jotka voi-

vat aiheuttaa reaaliaikaiselle sovellukselle hallitsemattomia prosessointipiikkejä. Projektin kannalta oli olennaista saada laaja tukiverkosto ja kirjastopohja, jotta jokaista pyörää ei tarvitsisi kehittää itse ja ongelmiin löytyisi muitakin motivoituneita kehittäjiä selvittämään niitä. Kyseisille kielille en löytynyt helposti tietokonegrafiikkaan liittyviä artikkeleita, aktiivisia kehittäjiä tai kirjastoja projektin toteutuksen ajanhetkellä. Vaatimuksien perusteella päädyin optimoituun ja hyvin tuettuun C++-kieleen. (44; 45; 46; 47.)

Kielen avulla voidaan toteuttaa hyvinkin skaalautuvia ja laskentatehokkaita sovelluksia. C++ mahdollistaa alustariippumattoman toteutuksen oikeanlaisten koodiratkaisujen ja kirjastojen avulla. C++ on erittäin mukautuva kieli ja tukee oliopohjaisia käsitteitä ja koodia sekä funktionaalisia ratkaisuja, jotka se on perinyt C-kielestä. C++:n joustavuudesta loistava esimerkki on sen ominaisuus, jolla voidaan upottaa Assembly-koodia suoraan käännettävään koodiin, jolloin voidaan yliajaa jopa kääntäjien toteutuksia optimaalisinta toteutusta varten.(45.)

6.5.2 Skriptikieli

Skriptikieliksi kilpailivat Javascript, LUA ja Python.Valitsin LUA:n edellisten hyvien kokemusten takia muissa projekteissa ja sen helpon integroimisen ja todella pienen muistijäljen ja tehokkuuden takia. (48; 44.)

LUA:n kääntäjä ja kielen määrittelyt voidaan helposti pitää projektin mukana. Esimerkiksi ne vievät vain 25 megatavua tilaa kovalevyltä omassa projektissani. (48.)

LUA:aa varten on toteutettu monia laajennuksia, kirjastoja ja rajapintoja moneen eri ohjelmointikieleen. Pienen muistijäljen ansiosta muistia ei tarvitse varata isoja määriä, ja se tapahtuu nopeasti. Sen ansiosta koodia voidaan vapauttaa ja varata helpostikin lennosta nopeaa ja dynaamista prototyyppittämistä varten. (48.)

6.5.3 Ohjelmointirajapinta

Grafiikkamoottorin eli C++-ohjelman ja asiakasohjelman eli LUA-ohjelman välille ohjelmoitava rajapinta toteutettiin LUABridge kirjaston avulla. LUABridge on avoimen koodin kirjasto, joka mahdollistaa oliokäsitteisen molemmin suuntaisen kommunikoinnin C++-

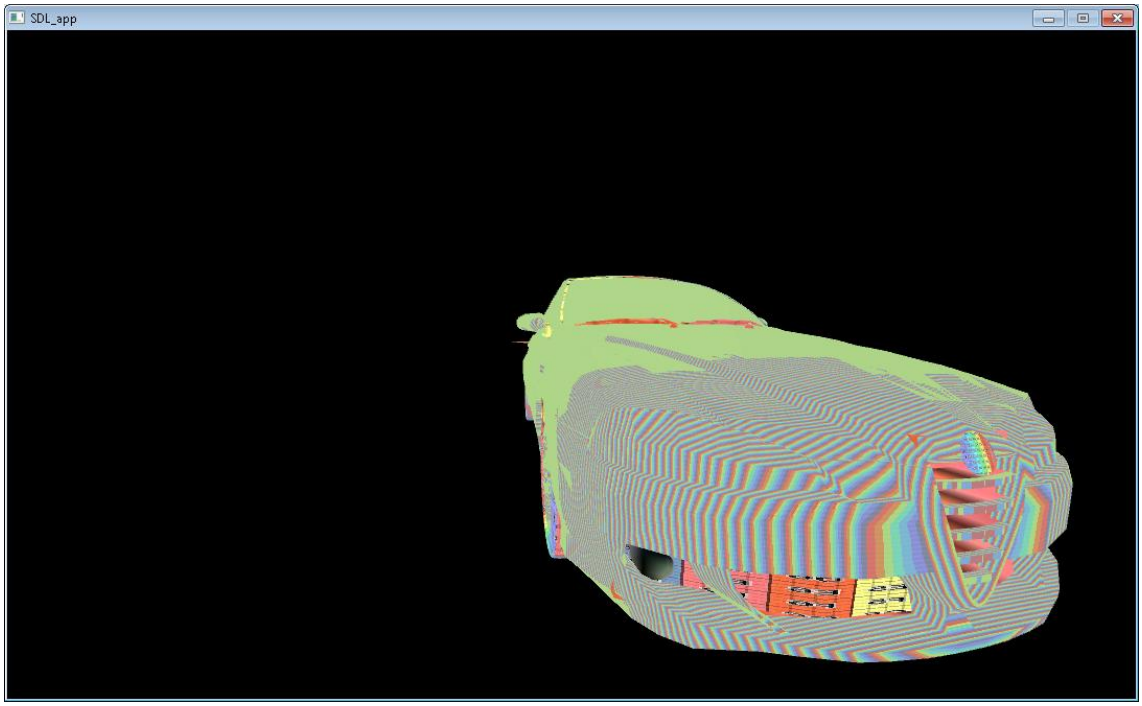
ja LUA-ohjelman välille. Sen avulla rajapinnan muodostaminen tapahtui erittäin siististi ja ristiriidattomasti, koska samoja oliokäsitteitä ja funktioita pystytään käyttämään samalla nimiavaruudella, vaikka ohjelmat ja ohjelmointikielet olivat erilaisia. Esimerkiksi perspektiiviin ja projektioon eli kameraan liittyvät tiedot lähetetään vektoreina asiakasohjelmalle pyynnön jälkeen.(48; 49.)

Ohjelmointirajapintaan pääsee käsiksi Lua-muuttujan rm kautta, joka löytyy LUA:n globaalista muistialueesta:

Kutsu	Selite
rm:initGeometry(modelPathString,texturePathString,vertexShaderPath fragmentShaderPath)	Kutsu alustaa piirrettävän geometrian. Kutsun ensimmäinen parametri on polku obj formaatissa olevaan mallitiedostoon, josta löytyy verteksit, tekstuurikoordinaatit ja verteksien normaalit, toinen parametri on polku tekstuuritiedostoon, kolmantena parametrina verteksisävytinohjelman polku ja neljäntenä parametrina fragmenttisävytinohjelman polku (formaatti GLSL).
rm:loadGeometries()	Kutsulla ladataan alustetut geometriat renderöitäviksi.
rm:translateGeometry(id,x,y,z)	Kutsulla liikutetaan geometriaa maailmakoordinaateissa. Parametreina ovat geometrian identifioiva järjestysluku ja liukuluku kutakin kolmiulotteista lukuavaruutta varten.
rm:scaleGeometry(id,x,y,z)	Kutsulla skaalataan geometriaa maailmakoordinaateissa. Parametreina ovat geometrian identifioiva järjestysluku ja liukuluku kutakin kolmiulotteista lukuavaruutta varten.
rm:rotateGeometry(id,x,y,z)	Kutsulla pyöritetään geometriaa maailmakoordinaateissa. Parametreina ovat geometrian identifioiva järjestysluku ja liukuluku kutakin kolmiulotteista lukuavaruutta varten.
rm:setLightPosGeometry(id,x,y,z)	Kutsulla asetetaan valonlähde geometrialle maailmakoordinaateissa. Parametreina ovat geometrian identifioiva järjestysluku ja liukuluku kutakin kolmiulotteista lukuavaruutta varten. Nykyinen versio tukee vain yhtä valonlähdettä.
rm:removeGeometry(id)	Kutsulla poistetaan geometria renderöimisprosessista.
rm:getCameraPosition()	Kutsulla pyydetään kameran sen hetkinen sijainti 3d vektorina.
rm:setCameraPosition(x,y,z)	Kutsulla asetetaan kameran sen hetkinen sijainti kolmen eri liukuluvun avulla.
rm:getCameraDirection()	Kutsulla pyydetään kameran sen hetkinen suunta 3d vektorina.
rm:setCameraDirection(x,y,z)	Kutsulla asetetaan kameran sen hetkinen suunta kolmen eri liukuluvun avulla.

Seuraavana on LUA-koodiesimerkki 3d-mallin auton lataamiseksi, joka on valaistettu sävytinohjelman määritellyn Phongin sävytinmallin mukaan. Kuviossa 18. on komentosarjan luoma kuva.

```
local objid2 = rm:initGeometry("Models/alfa147.obj","Textures/Asd.jpg",
"Shaders/VertexShader.transform", "Shaders/FragmentShader.color")
rm:loadGeometries()
rm:translateGeometry(objid2,-5.0,0.0,-5.0)
rm:scaleGeometry(objid2,0.01,0.01,0.01)
rm:rotateGeometry(objid2,-1.570796327,0,0)
```



Kuvio 18. Kuvankaappaus skriptin käskyttymän grafiikkamoottorin luomasta kuvasta.

6.5.4 Ikkunointi

Kuten edellisessä kuvassa hyvin näkyy, on ikkunointi tärkeä osa sovellusta. Ikkunan avulla abstrahoidaan moottorin luoman kuvan visualisointi käyttöjärjestelmää vasten. Ikkunan ominaisuuksia ovat resoluution eli pikseleiden suhde vaaka- ja pystysuunnassa määrittäminen, ikkunaan kohdistuvien tapahtumien kuunteleminen, kuvapuskurin ylimääräinen puskurointi, OpenGL-kontekstin hallinnointi.

Päädyin valitsemaan alustariippumattoman avoimen lähdekoodin kirjaston SDL (Simple DirectMedia Layer), joka tarjoaa yksinkertaisen matalan tason rajapinnan videontoistoon, äänentoistoon, näppäimistöön, hiireen, ikkunointiin sekä tiedostojen lukemiseen, lataamiseen ja kirjoittamiseen. Näiden kannalta ikkunointi, näppäimistön tietovirtojen ja kuvien lukeminen olivat olennaisia osia toteutusta varten. Valinta perustui omien ja muiden edellisten projektien hyviin kokemuksiin sekä kirjaston tarjoamaan hyvään dokumentaatioon ja tukiyhteisöön. (50.)

6.5.5 Resurssien lataaminen

Sovelluksessa resursseiksi määritellään tekstuurit, geometrioiden informaatio (verteksit, normaalit, tekstuurikoordinaatit), sävytinohjelmat ja skriptiohjelmat, joilla kehittäjä voi vaikuttaa ohjelman toimintaan.

Tekstuurien sovelluksen muistiin lataamiseen käytin SDL_image-lisäosakirjastoa, joka tukee suurta osaa yleisistä kuvaformaateista (BMP, GIF, JPEG, LBM, PCX, PNG, PNM, TGA, TIFF, WEBP, XCF, XPM, XV). MipMap-tekstuureja varten kirjoitin DDS-formaatin spesifikaatioiden perusteella jäsentelijän. Tekstuurit ladataan sovellukseen käyttöjärjestelmästä kehittäjän määrittämällä polulla, mutta on suositeltavaa pitää tekstuurit sovelluksen alikansiossa "Textures". Sovelluksen muistista tekstuurit ladataan näytönohjaimen muistiin, josta saadaan viite tekstuuripuskurin muistiosoitteeseen. Viitettä käytetään myöhemmin hyväksi renderöidessä geometriaa. (51.)

Geometrioiden dataa varten jouduin itse kirjoittamaan yksinkertaisen jäsentelijän Wavefront OBJ -tiedostoformaatin spesifikaatioiden mukaan. Jäsentelijä koostaa niistä indeksikohtaisia karttoja verteksien koordinaattien, normaalien ja tekstuurikoordinaattien perusteella. Jäsentelijä hakee OBJ-formaatissa olevan tiedoston sovellukseen käyttöjärjestelmästä kehittäjän määrittämällä polulla, mutta on suositeltavaa pitää ne sovelluksen alikansiossa "Models". (52.)

Sävytinohjelmat ladataan C++ STD-kirjaston luokkien fstream ja iostream tarjoamien funktioiden avulla sovelluksen muistiin. Käyttöjärjestelmän muistissa sijaitsevat tekstitiedostot ladataan näytönohjaimen muistiin, jossa ne käännetään ja tarkistutetaan GLSL-kääntäjän avulla. Virheettömän käännöksen jälkeen OpenGL palauttaa jokaista

sävytinohjelmaa kohden viitteen, jotka yhdistetään OpenGL:n ohjelmaolioon. Ohjelmaolion viitettä käytetään lopulta renderöidessä. (53.)

LUA-skriptitiedostot ladataan ajettaviksi LUA-kirjaston omalla kirjastokutsulla. Sovellus käyttää LUA-hakemiston alla olevaa main.lua tiedostoa alkupisteenä skriptien ajoon. Tiedosto ladataan kokonaisuudessaan muistiin. Tiedostossa tulee määrittää "initScene"-funktio, jota kutsutaan vain kerran moottorin käynnistyessä ja "update"-funktio, jota kutsutaan jokaisen moottorin päivittämän kuvan yhteydessä. (48.)

6.5.6 Resurssien hallinta

Lataamisen jälkeen tapahtuu resurssien esikäsittelyt ja hallinnointi. Esikäsittelyillä ja hallinnoinnilla vältetään turhaa prosessointia ja muistin varausta. Sovelluksessa on suunniteltu monta erilaista välimuistitasoa eri resursseille, joista voidaan viitata resursseihin lataamatta samaa resurssia uudelleen.

Tekstuureilla on varattu välimuisti, johon tallennetaan ajonaikaisesti tekstuuriolioita. Tekstuurioliot sisältävät erityyppisiä tekstuurikäsittelyjä erilaisille tekstuurityypeille.

Mallitiedot kartoitetaan välimuistiin tiedostopolun mukaan. Mallitiedoista otetaan talteen viitteet näytönohjaimessa sijaitseviin verteksien koordinaattien, normaaleiden ja tekstuurikoordinaattien puskureihin sekä erikseen verteksien lukumäärä.

Sävytinohjelmien ohjelmaolioiden viitteet on tallennettu linkitettyinä karttoina välimuistiin, joissa sävytinohjelmien tiedostopolut toimivat avaimina. Ohjelmaolioiden viitteet yhdistetään geometrioihin renderöintivaiheessa.

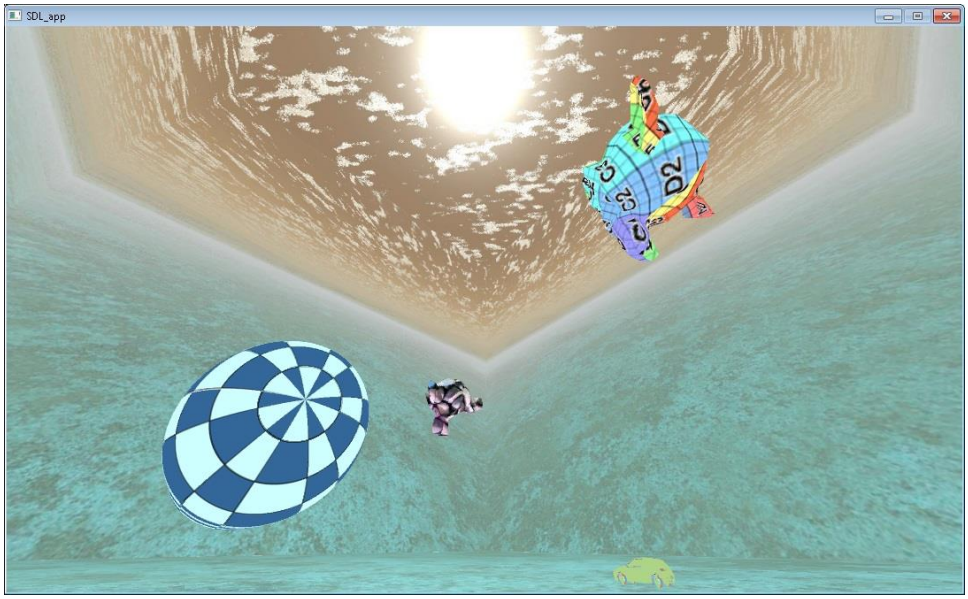
Kehittäjällä on mahdollisuus luoda LUA 5.1-version viitekehyksessä täysin oma arkkitehtuuri asiakasohjelmaan. Nopeaa prototyypittämistä varten kuitenkin suosittelen käyttämään LUA:n tarjoamaa "required"-funktiota, joka mahdollistaa koodimoduuleiden lataamisen ajettavaksi suoraan lennosta. Sen lisäksi suosittelen toteuttamaan staattisen tilanhallintafunktion, josta täysin dynaamiset funktiot voivat säilyttää ja hakea tilan tarpeen mukaan, koska täysin dynaamisesti ladattavan koodin kanssa tilanhallinnasta voi tulla haasteellista. (48.)

6.5.7 Näkyvyyden hallinta

Näkymää hallinnoidaan raa'asti optimoimatta C++ STD-kirjaston tarjoaman "vector" listaolion avulla. Geometriat tallennetaan ja lisätään listaan asiakasohjelman komentojen kautta. Sen lisäksi näkyvyyteen vaikuttaa OpenGL:n liukuhihna, jossa projektiomatriisin avulla leikataan näkyvät renderöitävät geometriat ja niiden primitiivit. (53.)

6.5.8 Kuvan piirtäminen

Kuvan muotojen ja värityksen tärkein käsite sovelluksen kannalta on geometriaolio, joka sisältää resurssien viitteet OpenGL:n sävytinohjelmiin, tekstuureihin, malleihin ja valaistukseen, kameraan ja matriiseihin, joiden avulla manipuloidaan näytönohjaimen liukuhihnan prosesseja. Kaikista sovelluksen hallinnoimista geometriaolioiden informaatiosta muodostetaan kokonaisuudessaan kolmiulotteinen näkymä, josta projisoidaan matriisien avulla näytönohjaimen kuvapuskuriin digitaalinen kuva OpenGL-kontekstiin. SDL ohjaa OpenGL-kontekstin kuvapuskurin oikeille päätelaitteille. Kuviossa 19. näkyy sovelluksen piirtämästä kuvasta. (5; 27; 50.)



Kuvio 19. Sovelluksen avulla luotu kuvanäkymä

Kamera on sovelluksen olio, joka hallitsee ja manipuloi projektio- ja näkymämatriiseja. Sen lisäksi kamera kommunikoi SDL:n kirjaston kanssa kuunnellakseen näppäimistön tietovirtoja ja toimii rajapintana kehittäjälle näkymän muokkaamista varten.

Projektiomatriisin avulla kartoitetaan kolmiulotteisen maailman geometrioiden pisteet kaksiulotteiseen kuvaan. Projektiomatriisilla vaikutetaan näkymän kuvasuhteeseen, näkökenttään, poimittavien arvojen minimi- ja maksimietäisyyksiin. (5; 27.)

Näkymämatriisin voidaan liikuttaa maailmassa nähtävien geometrioiden paikkaa ja suuntaa suhteessa kameran projektioon. Näkymämatriisin avulla illuusio siitä, että kamera liikkuu oikealle, mutta todellisuudessa maailmaa liikutetaan matriisin avulla vasemmalle. (5; 27.)

Mallimatriisi edustaa geometriaa maailmakoordinaatistossa. Sovelluksessa mallimatriisin toiminnallisuus ja tiedot on tallennettuna RenderGeometry-olioon. Mallimatriisi muodostuu kolmen eri matriisin tulosta translaatiomatriisista, rotaatiomatriisista ja skaalausmatriisista. Translaatiomatriisin avulla geometriaa liikutetaan maailmakoordinaatistossa. Rotaatiomatriisilla avulla geometriaa pyöritellään maailmakoordinaatiston avaruuksien suhteen. Skaalausmatriisilla geometrian verteksit skaalataan suhteessa maailmakoordinaatistoon. (5; 27.)

6.5.9 Kommunikaatio näytönohjaimen kanssa

Sovellus käyttää GL.h-header-tiedostoa saadakseen standardi OpenGL:n ominaisuudet käyttöönsä. Sen lisäksi sovellus tarvitsee glew.h-kirjastoa, jolla saadaan OpenGL 3.0-versiosta eteenpäin tulleet ominaisuudet aktivoitua käyttöön. Sen avulla aktivoin sävytinohjelmatuen GLSL 2.0 -versiolle ja erilaisia näytönohjaimen OpenGL puskurikomentoja.

OpenGL-rajapinnan kommunikaatio tapahtuu RenderGeometry-olion kautta. Olio lataa asiakasohjelman käskyttämät resurssit näytönohjaimen muistiin ja hallitsee niiden viitteitä. Olio lähettää näytönohjaimelle matriiseja, tekstuureita ja vektoreita, joita käytetään OpenGL-liukuhihnalla geometrioiden sävytykseen ja renderöimiseen.

Seuraavat nimetyt muuttujat lähetetään OpenGL:n liukuhihnaan.

Muuttuja	Selite
MVP	Projektiomatriisin, näkymämatriisin ja mallimatriisin tuloista muodostunut matriisi
V	Näkymämatriisi
M	Mallimatriisi
textureSampler	Viite tekstuuripuskuriin
lightPosition	Kolmiulotteinen vektori valaistuksen sijainnista

Muuttujiin voidaan viitata GLSL 2.0-kielen avulla sävytinohjelmissa seuraavanlaisesti :

```
uniform sampler2D textureSampler;
uniform mat4 MVP;
uniform mat4 V;
uniform mat4 M;
uniform vec3 lightPosition;
```

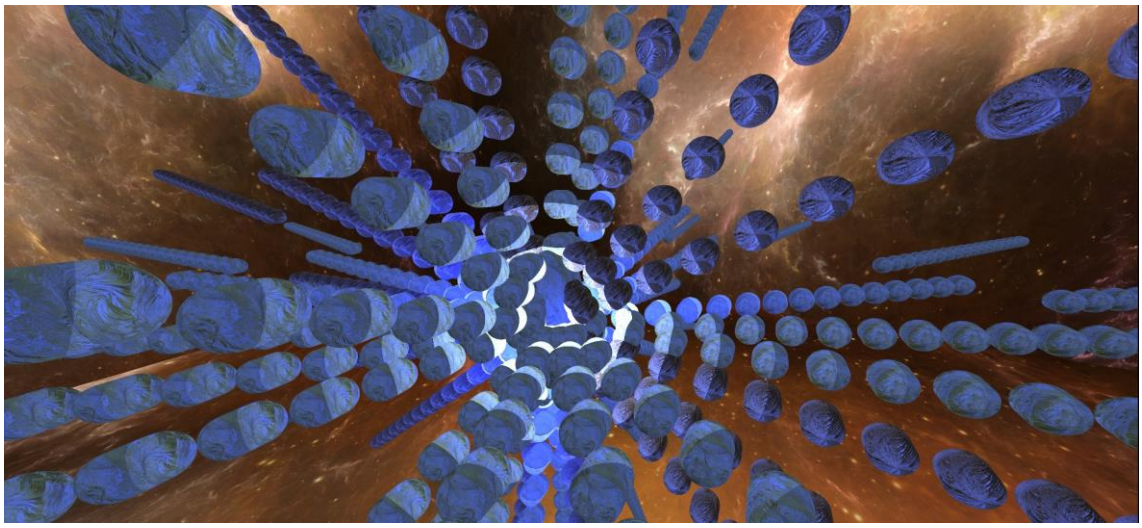
Sen lisäksi verteksien koordinaatit, tekstuurikoordinaatit ja normaalit lähetetään taulukkoina näytönohjaimelle. Taulukoihin voidaan viitata verteksisävytinohjelmassa seuraavasti. Sävytinohjelma interpoloi verteksin kerrallaan.

```
layout(location = 0) in vec3 vertexPosition_modelspace;
layout(location = 1) in vec2 vertexUV;
layout(location = 2) in vec3 vertexNormal_modelspace;
```

6.6 Sovelluksen matematiikka ja algoritmit

Sovelluksen matriisien ja vektorien tietorakenteen ylläpitämisen ulkoistin OpenGL Mathematics (GLM)-kirjastolle. Matriisien yleisimmät manipulaatiot kuten mallimatriisin avulla geometrioiden liikuttamisen, pyörittämisen ja skaalaamisen onnistuin toteuttamaan koulussa opitun matematiikan avulla. Kameran projektiomatriisin luomiseen käytin GLM:n tarjoamaa "perspective"-funktiota ja "lookAt"-funktiota näkymämatriisin luomiseen. (5; 27; 54.)

Sovellus ei itsessään ota kantaan minkäänlaisiin tietokonegrafiikan algoritmeihin vaan antaa työkalut kehittäjälle soveltaa grafiikkamoottorin ominaisuuksia omiin tarpeisiin. Kuviossa 20. on toteutettu matemaattinen animaatio, jossa pallot liikkuvat matemaattisten funktioiden mukaan.



Kuvio 20. Sovelluksella matemaattisesti muodostetun mallianimaation kuva

7 Jatkokehityssuunnitelmat

Projektinhallinta parantuisi huomattavasti integroimalla Biicode-ohjelma projektiin. Biicode vastaa ominaisuuksiltaan pelkistettyä versiota Javan jatkuvan integraation ohjelmaa Maven. Sen avulla voidaan toteuttaa erityisen hyvin ketterän kehityksen toimintamalleja. (55; 56.)

Uudemman version asentaminen kehitysympäristön mahdollistaa C++ 11-version ominaisuuksien käyttöönoton. (48.)

LUA 5.1 -version päivittäminen LUAJIT-versioksi. Monen eri LUAstate-kontekstin käynnistäminen ja niiden välisen kommunikoinnin toteuttaminen, jotta rinnakkaisesti toteutettava skriptikoodin prosessointi olisi mahdollista. (57.)

Eri tiedostojen malli- tai materiaaliformaattien jäsentelijöiden kirjoittaminen voi olla todella turhauttavaa välillä huonon tai salatun dokumentaation johdosta. Sitä varten on kehitetty Assimp-kirjasto, joka yhtenäistää monen eri malli- ja materiaalitiedostoformaatin käsittelyn. Sen integroiminen projektiin olisi erittäin tärkeää. (58.)

Turhien renderöintikomentojen välttämiseksi olisi välttämätöntä toteuttaa poiminta algoritmi. Se voisi olla esimerkiksi rajageometrioiden volumetrisiin arvoihin perustuva algoritmi. (59.)

Kuvan piirtämisen tehokkuuden parantamiseksi ja parempien ominaisuuksien saavuttamiseksi tulisi ohjelman tukea uusia näytönohjainrajapintoja. Khronos-ryhmittymän Vulkan rajapinta vaikuttaa erittäin lupaavalta tehokkaalta tavalta renderöidä sarjallisesti geometrioita. OpenCL:n rinnakkaisen renderöimisen toteuttaminen kuulostaa myös mielenkiintoiselta, jolloin laskentakapasiteettia vaativien algoritmien toteuttaminen olisi mahdollista. (60.)

8 Oppimien asioiden analysointi ja arviointi

Työn aikana olen kehittynyt ohjelmistoprojektin suunnittelemisessa, aikatauluttamisessa, ja hallinnoimisessa. Nämä aspektit olivat erittäin haastavia hoitaa kaikkien muiden projektien ja töiden lomassa. Mielestäni pääsin hyvin projektin asettamiin tavoitteisiin, jotka asetettiin rajauksien kautta.

Työn aikana informaation hankinta- ja suodattamistaidot edistyivät huomattavasti. Internetin järjettömän suureen tietomäärään on helppo hukkua. Itselleni tärkeimmiksi tiedonlähteiksi muodostuivat tieteelliset dokumentit ja tutkimukset, aiheeseen liittyvien sovelluksien dokumentaatiot, Wikipedia ja isojen laitevalmistajien nettisivut ja keskustelupalstat.

Ohjelmointikielissä taito- sekä tietotasot nousivat huomattavasti C++ ja LUA suhteen toteuttaessani grafiikkamoottoria. Monet eri tiedonkäsittelyalgoritmit ja -paradigmat tulivat tutuiksi optimoidessani ja toteuttaessani koodia.

Matemaattiset taidot ja tiedot harjaantuivat geometrioita, vektoreita ja matriiseja manipuloivissa toiminnollisuuksissa.

Integraatiotaidot kehittyivät valtavasti erilaisten rajapintojen ja kirjastojen käyttöönoton yhteydessä.

Projektin dokumentaatiosta tuli itselleni suurin huolenaihe. Olen hyvin vahvoilla, kun pääsen toteuttamaan ja miettimään ratkaisuja monimutkaisimpiin ongelmiin, mutta olen todella hidas symbolisoimaan ja selittämään ratkaisuja muille ihmisille kirjoittaen. Sen suhteen olen varmasti parantunut suhteellisesti suurimman määrän, vaikka vielä pidänkin dokumentointia suurena heikkoutenani.

Kaikkia opittuja taitoja on edes vaikea hahmottaa, mutta tunnen kokemuksen antaneen itselleni paljon työkaluja tuleviin projekteihin ja varmuutta toimia ohjelmistoteknillisessä ympäristössä

Lähteet

1 Game engine, http://en.wikipedia.org/wiki/Game_engine. Luettu 10.04.2015.

2 Software engine, http://en.wikipedia.org/wiki/Software_engine. Luettu 10.04.2015.

3 Computer graphics (computer science), http://en.wikipedia.org/wiki/Computer_graphics_%28computer_science%29. Luettu 10.04.2015.

4 3D computer graphics, http://en.wikipedia.org/wiki/3D_computer_graphics. Luettu 10.04.2015.

5 Samuel R. Buss, 3-D computer graphics: a mathematical introduction with OpenGL.

6 Computer-aided design, http://en.wikipedia.org/wiki/Computer-aided_design. Luettu 10.04.2015.

7 Scientific visualization, http://en.wikipedia.org/wiki/Scientific_visualization. Luettu 10.04.2015.

8 Virtual reality, http://en.wikipedia.org/wiki/Virtual_reality. Luettu 10.04.2015.

9 Augmented reality, http://en.wikipedia.org/wiki/Augmented_reality. Luettu 10.04.2015.

10 2014 NVIDIA Corporation, What is GPU Computing? , http://www.nvidia.com/object/what-is-gpu-computing.html. Luettu 10.04.2015.

11 Sangjin Han, On Fair Comparison between CPU and GPU, http://www.eecs.berkeley.edu/~sangjin/2013/02/12/CPU-GPU-comparison.html

12 Victor W Lee, Changkyu Kim, Jatin Chhugani, Michael Deisher, Daehyun Kim, Anthony D. Nguyen, Nadathur Satish, Mikhail Smelyanskiy,Srinivas Chennupaty, Per Hammarlund, Ronak Singhal, Pradeep Dubey, Debunking the 100X GPU vs. CPU Myth: An Evaluation of Throughput Computing on CPU and GPU, http://www.eecs.berkeley.edu/~sangjin/static/pub/nsdi2011_sslshader.pdf. Luettu 10.04.2015.

13 2014 NVIDIA Corporation, CUDA Parallel Computing , http://www.nvidia.co.uk/object/cuda-parallel-computing-uk.html. Luettu 10.04.2015.

14 2014 HSA Foundation, THE HETEROGENEOUS SYSTEM ARCHITECTURE ITS (NOT) ALL ABOUT THE GPU Inc, http://amd-dev.wpengine.netdnacdn.com/wordpress/media/2012/10/keynote_theHSA_itsnotallabouttheGPU.pdfee.pdf. Luettu 10.04.2015.

15 OpenGL, https://www.opengl.org/wiki/Main_Page. Luettu 10.04.2015.

16 Direct3D, http://en.wikipedia.org/wiki/Direct3D. Luettu 10.04.2015.

17 OpenGL, http://en.wikipedia.org/wiki/OpenGL. Luettu 10.04.2015.

18 What is D3D, https://msdn.microsoft.com/en-us/library/windows/desktop/hh769064%28v=vs.85%29.aspx#What_is_D3D. Luettu 10.04.2015.

19 High-Level Shading Language, http://en.wikipedia.org/wiki/High-Level_Shading_Language. Luettu 10.04.2015.

20 Tesselation(computer graphics) http://en.wikipedia.org/wiki/Tessellation_%28computer_graphics%29. Luettu 10.04.2015.

21 Displacement mapping, http://en.wikipedia.org/wiki/Displacement_mapping. Luettu 10.04.2015.

22 Pipeline render, http://www.cs.kent.edu/~farrell/graphics/opengl/openglprog1.2/opengl/render.html. Luettu 10.04.2015.

23 Rasterisation, http://en.wikipedia.org/wiki/Rasterisation. Luettu 10.04.2015.

24 OpenCL, http://en.wikipedia.org/wiki/OpenCL. Luettu 10.04.2015.

25 OpenCL, https://www.khronos.org/opencl/. Luettu 10.04.2015.

26 CUDA, http://en.wikipedia.org/wiki/CUDA. Luettu 10.04.2015.

27 Steve Cunningham , Computer graphics : programming in OpenGL for visual communication.

28 Phong shading, http://en.wikipedia.org/wiki/Phong_shading. Luettu 10.04.2015.

29 Bump mapping, http://en.wikipedia.org/wiki/Bump_mapping. Luettu 10.04.2015.

30 Normal mapping,http://en.wikipedia.org/wiki/Normal_mapping. Luettu 10.04.2015.

31 Shadow mapping, http://en.wikipedia.org/wiki/Shadow_mapping. Luettu 10.04.2015.

32 Ambient occlusion, http://en.wikipedia.org/wiki/Ambient_occlusion. Luettu 10.04.2015.

33 Mipmap, http://en.wikipedia.org/wiki/Mipmap. Luettu 10.04.2015.

34 Ray tracing graphics, http://en.wikipedia.org/wiki/Ray_tracing_%28graphics%29. Luettu 10.04.2015.

35 Radiosity (computer graphics), http://en.wikipedia.org/wiki/Radiosity_(computer_graphics). Luettu 10.04.2015.

36 Ogre3d website and documentation, http://ogre3d.org/. Luettu 10.04.2015.

37 OGRE, http://en.wikipedia.org/wiki/OGRE. Luettu 10.04.2015.

38 Horde3d website and documentation, http://horde3d.org/. Luettu 10.04.2015.

39 Horde3D, http://en.wikipedia.org/wiki/Horde3D. Luettu 10.04.2015.

40 POV-Ray, http://en.wikipedia.org/wiki/POV-Ray. Luettu 10.04.2015.

41 POV-Ray website, http://www.povray.org/. Luettu 10.04.2015.

42 Microsoft Visual Studio, http://en.wikipedia.org/wiki/Microsoft_Visual_Studio. Luettu 10.04.2015.

43 Git, http://www.git-scm.com/. Luettu 10.04.2015.

44 Programming language benchmarks, http://benchmarksgame.alioth.debian.org/. Luettu 10.04.2015.

45 C++, http://en.wikipedia.org/wiki/C%2B%2B. Luettu 10.04.2015.

46 Java, http://en.wikipedia.org/wiki/Java. Luettu 10.04.2015.

47 Python (programming language)http://en.wikipedia.org/wiki/Python_(programming_language). Luettu 10.04.2015.

48 Lua website and documentation, http://www.lua.org/. Luettu 10.04.2015.

49 Source code and documentation, https://github.com/vinniefalco/LuaBridge. Luettu 10.04.2015.

50 SDL documentation, https://www.libsdl.org/. Luettu 10.04.2015

51 SDL_image ,https://www.libsdl.org/projects/SDL_image/. Luettu 10.04.2015.

52 Wavefront .obf file, http://en.wikipedia.org/wiki/Wavefront_.obj_file. Luettu 10.04.2015.

53 C++ references, http://www.cplusplus.com/reference/. Luettu 10.04.2015.

54 GLM website and documentation, http://glm.g-truc.net/0.9.6/index.html. Luettu 10.04.2015.

55 Biicode website and documentation, http://docs.biicode.com/. Luettu 10.04.2015.

56 Apache Maven, http://en.wikipedia.org/wiki/Apache_Maven. Luettu 10.04.2015.

57 LuaJIT website and documentation, http://luajit.org/. Luettu 10.04.2015.

58 Assimp, http://assimp.sourceforge.net/. Luettu 10.04.2015.

59 GPU Gems Chapter 29, http://http.developer.nvidia.com/GPUGems/gpugems_ch29.html. Luettu 10.04.2015.

60 Vulkan, https://www.khronos.org/vulkan. Luettu 10.04.2015.

61 http://en.wikipedia.org/wiki/Phong_shading#/media/File:Phong_components_version_4.png

62 http://en.wikipedia.org/wiki/Bump_mapping#/media/File:Bump-map-demo-full.png

63 http://en.wikipedia.org/wiki/Normal_mapping#/media/File:Normal_map_example.png

64 http://en.wikipedia.org/wiki/Normal_mapping#/media/File:Normal_map_example_with_scene_and_result.png

65 http://en.wikipedia.org/wiki/Shadow_mapping#/media/File:7fin.png

66 http://en.wikipedia.org/wiki/Ambient_occlusion#/media/File:Show_how_3D_real_time_ambient_occlusion_works_2013-11-23_10-45.jpeg

67 http://en.wikipedia.org/wiki/Mipmap#/media/File:MipMap_Example_STS101.jpg

68 http://en.wikipedia.org/wiki/Ray_tracing_%28graphics%29#/media/File:Recursive_raytrace_of_a_sphere.png

69 http://upload.wikimedia.org/wikipedia/commons/5/55/Radiosity_Comparison.jpg

70 http://www.ogre3d.org/

71 http://fi.wikipedia.org/wiki/OGRE#/media/File:OGRE_screenshot_01.png

72 http://upload.wikimedia.org/wikipedia/commons/thumb/e/ec/Glasses_800_edit.png/1280px-Glasses_800_edit.png

73 http://www.horde3d.org/screenshots/chicago.jpg

74 http://en.wikipedia.org/wiki/Displacement_mapping#/media/File:Displacement.jpg

75 http://commons.wikimedia.org/wiki/File:The_OpenGL_-_DirectX_graphics_pipeline.png